Grundlagen Rechnernetze und Verteilte Systeme

SoSe 2013

Kapitel 5: Sitzungs-, Darstellungs- und Anwendungsschicht

Prof. Dr.-Ing. Georg Carle

Dipl.-Ing. Stephan M. Günther, M.Sc.
Nadine Herold, M.Sc.
Dipl.-Inf. Stephan Posselt

Fakultät für Informatik
Lehrstuhl für Netzarchitekturen und Netzdienste
Technische Universität München

Worum geht es in diesem Kapitel?

1 Motivation

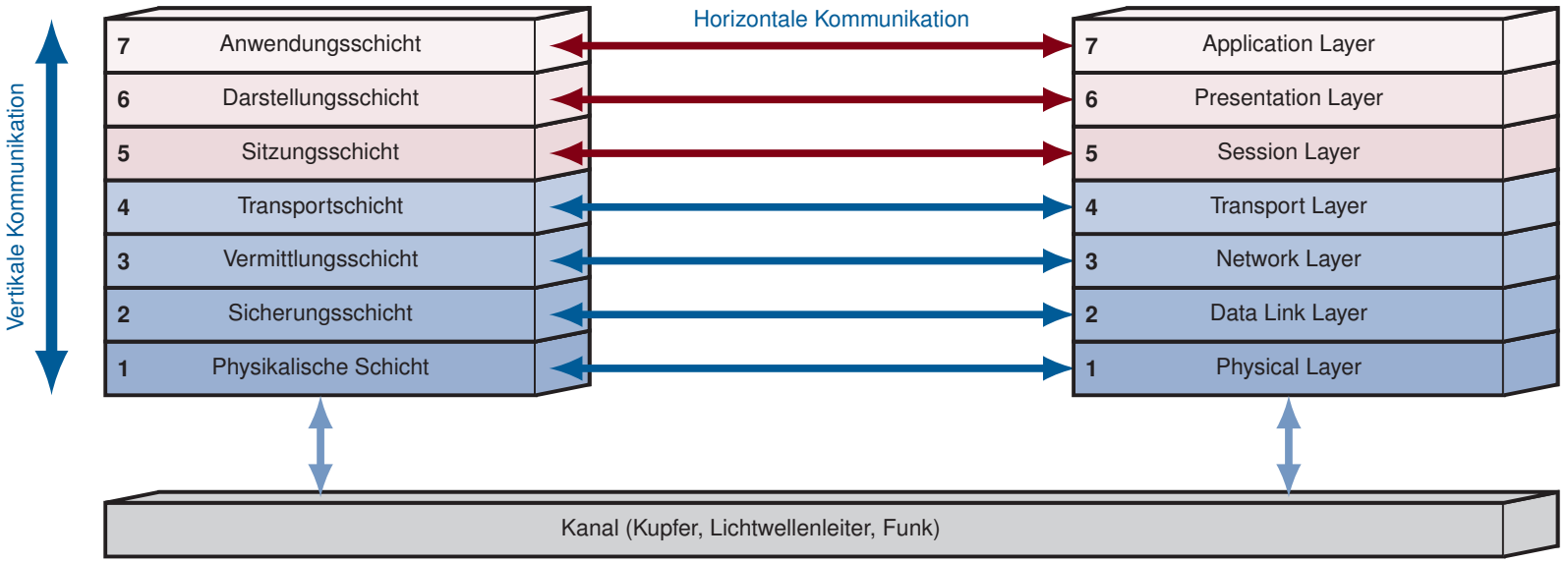
2 Sitzungsschicht

3 Einschub: Kryptographie und Netzsicherheit

4 Darstellungsschicht

5 Anwendungsschicht

Einordnung im ISO/OSI-Modell



Modell und Realität

- Dienste der Sitzungs- und der Darstellungsschicht sind in einzelnen Fällen in Form standardisierter Protokolle implementiert.
- In anderen Fällen sind Funktionen, die der Sitzungs- bzw. der Darstellungsschicht zuzuordnen sind, in die Anwendung integriert.
- Die Standards der ITU-Serie X.200 beschreiben Dienste der sieben OSI-Schichten, sowie Protokolle zur Erbringung dieser Dienste. Die in diesen Standards vorgenommene Strukturierung ist nützlich. Etliche OSI-Protokolle haben in der Praxis kaum Bedeutung.

Im Folgenden werden wir

- die Aufgaben der Sitzungs- und Darstellungsschicht erläutern,
- Kryptografischen Grundlagen vorstellen, und
- beispielhaft einige Protokolle kennenlernen, welche den Schichten 5 und 6 zugeordnet werden können.

Im Anschluss lernen wir einige häufig verwendete Protokolle der Anwendungsschicht kennen.

Übersicht

Sitzungsschicht

Die Sitzungsschicht kann nach X.200 in zwei verschiedenen **Modi** betrieben werden:

- **Connection-Oriented Mode:**
  Die Sitzungsschicht baut eine Verbindung zwischen den Kommunikationspartnern auf, die über die Dauer einzelner Datentransfers hinweg erhalten bleibt.
  Es werden die Phasen Verbindungsaufbau, Datentransfer und Verbindungsabbau unterschieden.
  Die Sitzungsschicht kann in diesem Fall vielfältige Aufgaben erfüllen.
  Die unterstützte Funktionalität beinhaltet die Koordination mehrerer beteiligter Parteien bezüglich Datenfluss, verwendeter Dienste, Aushandlung diverser Parameter für den Kommunikationsablauf und Identifikation der Verbindungen.

- **Connectionless Mode:**
  In diesem Fall wird durch die Sitzungsschicht nur ein sehr einfacher Dienst erbracht.
  Von der dienstnehmenden Schicht werden die zu übertragenden Daten entgegengenommen, zusammen mit Informationen zur Adressierung und zu den Dienstgüteanforderungen, und an die Transportschicht weitergereicht.

Eine Verbindung der Sitzungsschicht ist nicht gleichbedeutend mit einer Verbindung der Transportschicht! Eine Session kann beispielsweise nacheinander mehrere TCP-Verbindungen beinhalten.

Dienste der Sitzungsschicht

Definition (Session)

Eine Session beschreibt die Kommunikation zwischen zwei oder mehreren Teilnehmern, mit definiertem Anfang und Ende und sich daraus ergebender Dauer.

Um für die dienstnehmende Schicht (Darstellungsschicht) eine Dialogführung zu ermöglichen, müssen gegebenenfalls mehrere Transportschicht-Verbindungen verwendet und kontrolliert werden. Dies kann auch die Behandlung abgebrochener und wiederaufgenommer TCP-Verbindungen beinhalten.
Im Connection-Oriented Mode werden verschiedene Dienste angeboten:

- **Aufbau** und **Abbau** von Sessions,
- normaler und beschleunigter **Datentransfer** [1],
- Token-Management zur **Koordination** der Teilnehmer,
- **Synchronisation** und Resynchronisation,
- **Fehlermeldungen** und Aktivitätsmanagement, sowie
- **Erhaltung** und **Wiederaufnahme** von Sessions nach Verbindungsabbrüchen.

Die Sitzungsschicht stellt im Connectionless Mode folgenden einfachen Dienst bereit:

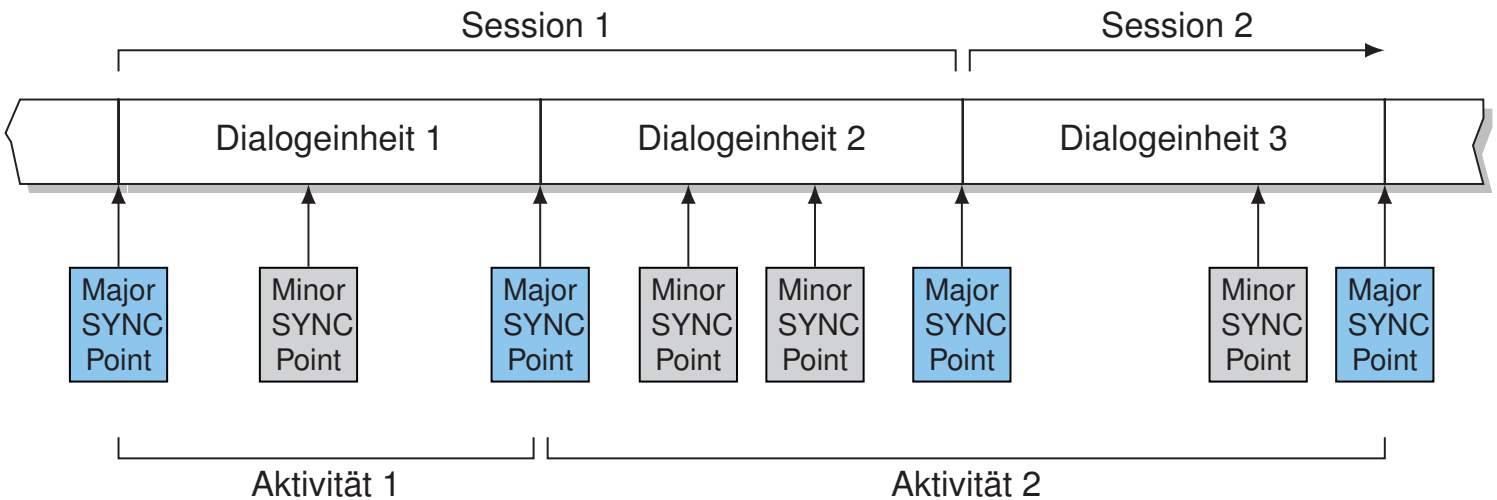
- **Datentransfer**

[1] Expedited Data Transfer: dringliche Daten, wie Alarme oder Interrupts

Funktionseinheiten der Sitzungsschicht

Die Dienste der Sitzungsschicht werden gemäß dem Standard X.215 in Funktionseinheiten gruppiert:

Bezeichnung	Beschreibung
Kernel	Bereitstellung von Basisfunktionen
Halb-duplex	Abwechselndes Senden und Empfangen
Duplex	Gleichzeitiges Senden und Empfangen
Negotiated Release	Beenden der Session mit gegenseitigem Bestätigen
Expedit Data	Beschleunigter Datentransfer
Activity Management	Logische Strukturierung der Session
Major Synchronization	Interne Strukturierung der Sessions in Dialogeinheiten
Minor Synchronization	Interne Strukturierung der Dialogeinheiten



Kombination von Funktionseinheiten

Unterschied zwischen Aktivitätsmanagement und Synchronisation

Aktivitäten bestehen aus mehreren Dialogeinheiten und können jederzeit unterbrochen und später (in der gleichen oder auch einer anderen Sitzung) wieder aufgenommen werden; d.h. eine Aktivität kann auch über mehrere Sessions hinweg existieren.
Synchronisation erfolgt durch sog. Synchronisationspunkte, an welchen Sessions oder Aktivitäten wieder aufgesetzt werden können, oder an welchen eine Resynchronisation (Zurücksetzen) möglich ist. Die Kommunikationspartner können hier prüfen, wie weit die Kommunikation, z. B. eine Datenübertragung, vorangeschritten ist.

Diese Funktionseinheiten dürfen allerdings nur in bestimmten Kombinationen verwendet werden, um Inkompatibilitäten zu vermeiden:

- BCS Basic Combined Subset
  Kernel, Halb-Duplex/Duplex
- BSS Basic Synchronized Subset
  Kernel, Halb-Duplex, Negotiated Release, Minor/Major synchronize, Resynchronize
- BAS Basic Activity Subset
  Kernel, Halb-Duplex, Minor sychronize, Execptions, Activity Management

Eine gültige Kombination wird vor Beginn der Session zwischen den Kommunikationspartnern ausgehandelt.

Synchronisation

Es werden folgende Arten von Synchronisationspunkten unterschieden:

- Major Synchronisationspunkte
- Minor Synchronisationspunkte

Definition (Major und Minor Synchronisationspunkte)

Major Synchronisationspunkte werden verwendet, um die Struktur der ausgetauschten Daten in eine Serie von Dialogeinheiten zu zerlegen. Sie müssen explizit bestätigt werden.

Minor Synchronisationspunkte werden für die Strukturierung innerhalb dieser Dialogeinheiten verwendet. Sie können bestätigt werden. Bis zu einem solchen Punkt versandte Daten werden nicht von einem Resynchronisationsprozess verworfen.

Um die Kommunikation zu steuern, werden Marken (Token) verwendet:

- Datenmarken (“data token” - geben an, wer bei der Verwendung des Halb-duplex-Betriebs senden darf)
- Beendigungsmarken (“release token” - beenden eine Sitzung)
- Synchronisationsmarken (“synchronize-minor token”)
- Aktivitätsmarken (“activity-major token”)

Quality of Service im Session Layer

Auf der Sitzungsschicht kann zwischen verschiedenen QoS-Parametern unterschieden werden:

- Service Parameter
- Performance Parameter

Es werden zunächst die verschiedenen Service Parameter erläutert.

Definition (Service Parameter)

Protection: Beschreibt den Schutz gegen unautorisiertes Monitoring oder Manipulation von Nutzerinformationen

Priorität: Beschreibt die Relation zwischen den einzelnen Sessions (Aufteilung von Ressourcen)

Resilience: Beschreibt die Wahrscheinlichkeit, dass eine Session ordnungsgemäß durchgeführt werden kann bzw. wie viele Fehler auftreten dürfen

Performance Parameter des Session Layers

- Aufbau der Session: Establishment Delay und Establishment Failure Probability
- Datentransfer
  - Durchsatz und Transit Delay
  - Residual Error Rate und Transfer Failure Probability
- Abbau der Session: Release Delay und Release Failure Probability

Definition (Residual Error Rate)

Die RER bezeichnet das Verhältnis der Menge falsch übertragener, verlorener oder dupliziert empfangener Daten zur Gesamtmenge der gesendeten Daten:

$$\text{RER} = \frac{S_e + S_l + S_x}{S}.$$

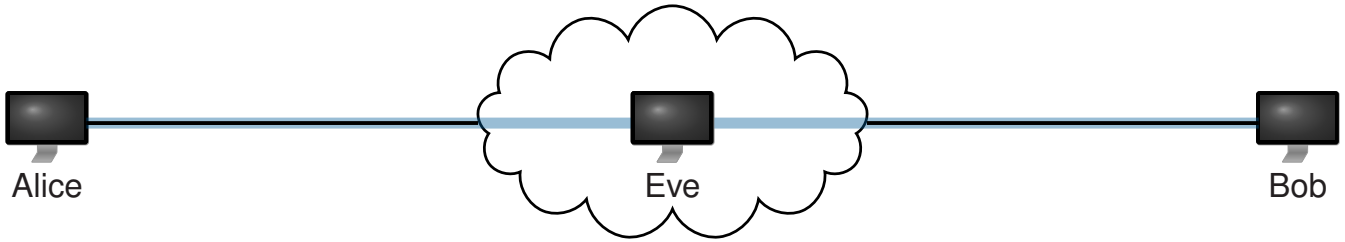
- S_e beschreibt falsch übertragene Daten, S_l verloren gegangene Daten und S_x dupliziert empfangenen Daten.
- S beschreibt die Gesamtmenge gesendeter Daten.
- Die Gesamtmenge S gesendeter Daten setzt sich zusammen aus $S_s + S_e + S_l + S_x$, mit der Menge korrekt übertragener Daten S_s.
- Die Menge empfangener Daten setzt sich zusammen aus $S_s + S_e + S_x$.

Einschub: Kryptographie

Einschub: Kryptographie

Problemstellung:

- Alice und Bob wollen miteinander kommunizieren.



- Alice hat keinen Einfluss darauf, wie Datenpakete durch das Internet geleitet werden, wer sie mitlesen oder sogar modifizieren kann.
- Alice kann sich (trotz korrekter Adressierung von Bob) nicht einmal sicher sein, überhaupt mit Bob zu kommunzieren:
  - Der Angreifer Eve könnte sich sowohl Alice als auch Bob gegenüber als der jeweils andere Kommunikationspartner ausgeben.
  - Einzige Voraussetzung dafür ist, dass Eve die Pakete von Alice und Bob „abfangen“ kann.
  - Befindet sich Eve im lokalen Netz von Alice oder Bob, ist das sehr einfach: ARP-Spoofing.

Idee: Alice und Bob authentisieren sich gegenseitig und verschlüsseln Ihre Daten, so dass

- sichergestellt ist, dass Alice und Bob auch wirklich diejenigen sind, die sie vorgeben zu sein, und
- nur der jeweils andere Kommunikationspartner in der Lage ist, die Daten wieder zu entschlüsseln.

Ziele kryptografischer Verfahren

Mittels kryptographischer Verfahren werden eine Reihe von Zielen verfolgt:

- Integrität
  Bob will sich sicher sein, dass die Daten von Alice auf dem Weg nicht verändert wurden.
- Authentizität
  Bob will sich sicher sein, dass die Daten auch wirklich von Alice stammen und nicht von jemadem, der sich für Alice ausgibt.
- Vertraulichkeit
  Es soll verhindert werden, dass unbefugte Dritte Nachrichten zwischen Alice und Bob mitlesen können.
- Verbindlichkeit / Nichtabstreitbarkeit
  Dem Sender einer Nachricht soll es nicht möglich sein zu bestreiten, dass er der Urheber einer bestimmten Nachricht ist.

Diese Ziele werden i. d. R. nur durch das (komplizierte) Zusammenspiel aus

- Sitzungsprotokollen
- Schlüsselaustauschverfahren bzw. -protokollen,
- Verschlüsselungsalgorithmen und
- Hashverfahren

erreicht.

Klassifizierung von Verschlüsselungsverfahren

Grundsätzlich unterscheidet man:

- **Symmetrische Verfahren**
  - Sender und Empfänger einigen sich auf ein gemeinsames Geheimnis (engl. Shared Secret, ugs. „Passwort“oder „Schlüssel“).
  - Mittels dieses Schlüssels können Daten verschlüsselt und auch wieder entschlüsselt werden (daher „symmetrische“ Verfahren).
- **Asymmetrische Verfahren**
  - Alice und Bob besitzen jeweils zwei Schlüssel: Einen Public Key und einen Private Key.
  - Der öffentliche Schlüssel ist prinzipiell jedem zugänglich, der private Schlüssel wird um jeden Preis geheim gehalten.
  - Alice verschlüsselt eine Nachricht mit dem öffentlichen Schlüssel von Bob.
  - Zur Entschlüsselung ist der passende private Schlüssel notwendig, den nur Bob besitzt.
  - Dieses Prinzip steht hinter den RSA-Schlüsseln die Sie zum Zugriff auf Ihre GRNVS-VMs benötigen.

Im Folgenden betrachten wir zunächst ein symmetrisches Verschlüsselungsverfahren:

- RC4 (Rivest Cipher 4) als symmetrisches Verschlüsselungsverfahren in Form einer Stromchiffre, in Verbindung mit dem
- Diffie-Hellman-Schlüsselaustauschverfahren, veröffentlicht von Whitfield Diffie und Martin Hellman (Stanford University) in IEEE Transactions on Information Theory, Vol. 22, Nr. 6, S. 644–654, 1976.

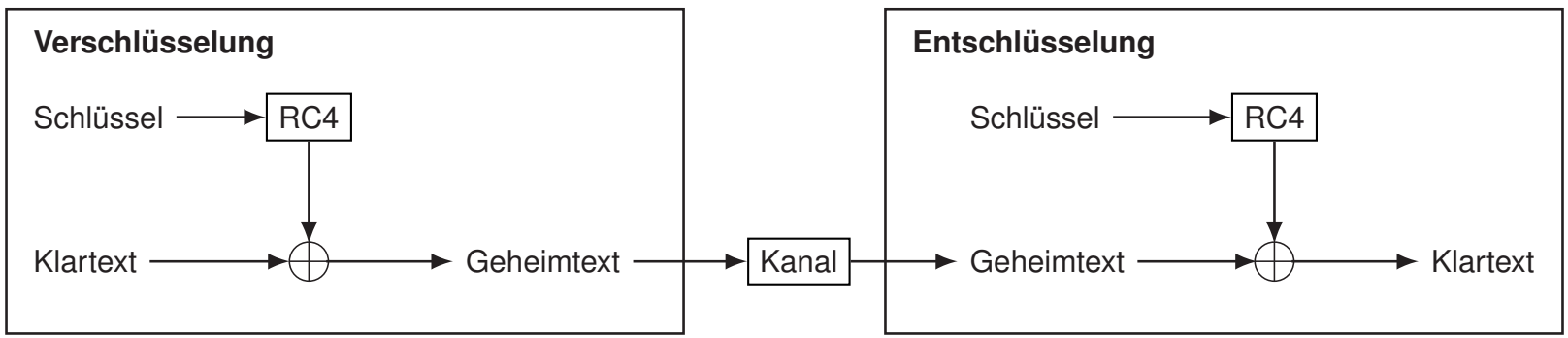
Symmetrische Verfahren: RC4

Verschlüsselung:

- Der Sender erzeugt einen mit Hilfe eines gemeinsamen Geheimnisses einen Schlüsselstrom (engl. Cipher-Stream).
- Dies ist ein pseudozufälliger Bitstrom. Mit Hilfe des gemeinsamen Geheimnisses kann dieser Bitstrom von beiden Kommunikationspartnern reproduziert werden.
- Der Schlüsselstrom wird mit der zu verschlüsselnden Nachricht (Klartext, engl. Plain Text) bitweise XOR verknüpft.
- Der resultierende Datenstrom wird als Geheimtext (engl. Cipher Text) bezeichnet.

Entschlüsselung:

- Der Empfänger kann mit Hilfe des gemeinsamen Geheimnisses denselben Schlüsselstrom erzeugen.
- XOR-Verknüpfung aus Schlüsselstrom und Geheimtext ergibt wieder den Klartext.



Wie erzeugt RC4 den Schlüsselstrom?

- RC4 initialisiert in Abhängigkeit des Schlüssels eine sog. S-Box (Substitutions-Box), welche 256 Byte fasst mit den Zahlen 0 bis 255.
- Im Anschluss werden in jedem Schritt zwei Zahlen aus der S-Box permutiert und die Summe der beiden permutierten Zahlen modulo 256 als Index zu einer neuen Zahl in der S-Box verwendet, welche als „Zufallszahl“ zurückgegeben wird.[2]

Bei RC4 ist zu beachten:

- Es dürfen niemals zwei Nachrichten mit demselben Schlüsselstrom verschlüsselt werden, da andernfalls ein Angreifer, dem eine Nachricht im Klartext bekannt ist, auch die andere Nachricht im Klartext berechnen kann.
- Aufgrund der einfachen Funktionsweise lassen die ersten Bytes des Schlüsselstroms Rückschlüsse auf den Schlüssel selbst zu und sollten daher verworfen werden.
- Wie jeder Generator für pseudozufällige Zahlen hat RC4 eine begrenzte Periodendauer, d. h. irgendwann wiederholt sich der Schlüsselstrom.
- Bei RC4 wurden bestimmte Korrelationen zwischen Schlüssel und Schlüsselstrom gefunden, so dass RC4 heute als nicht mehr sicher angesehen wird.

Verschlüsselungsprotokolle (Cipher Suites), die RC4 nutzen:

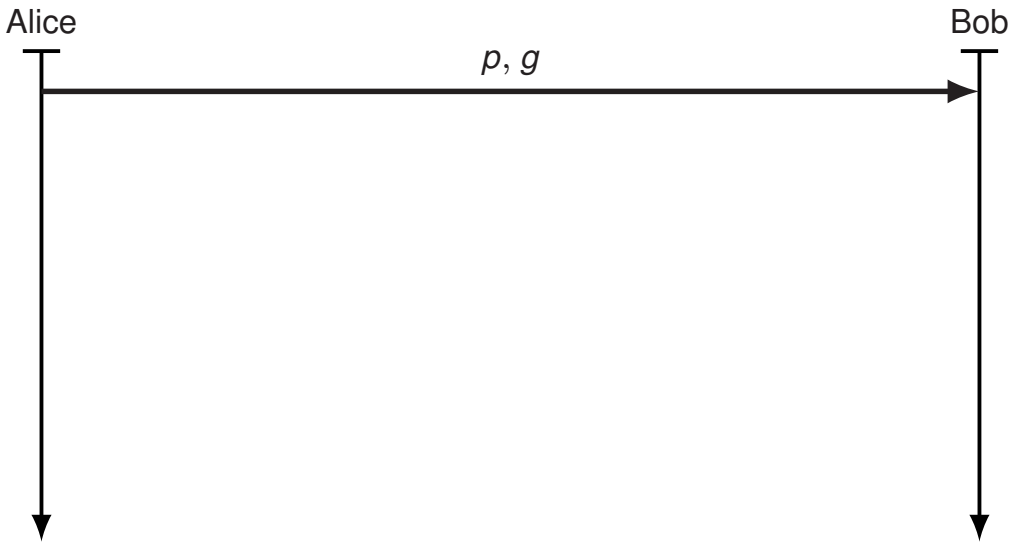
- WEP[3] (Wired Equivalent Privacy)
- WPA-TKIP (WiFi Protected Access Temporal Key Integrity Protocol)
- Als optionaler Algorithmus in IPSec, SSL/TLS, SSH, Kerberos

[2] Diese Darstellungsweise ist etwas vereinfachend.

[3] Die Schwäche liegt hier nicht bei RC4 selbst sondern bei der Art, wie RC4 genutzt wird.

Wie tauschen Sender und Empfänger die gemeinsamen Schlüssel aus?

Eine Möglichkeit besteht in der Nutzung des Diffie-Helmann-Verfahrens:

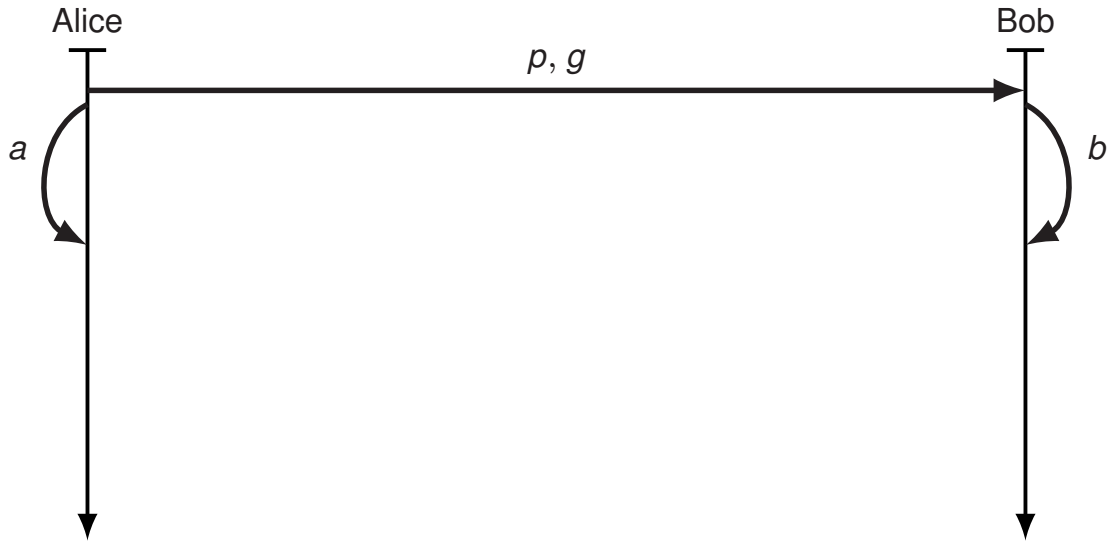


- Alice und Bob einigen sich zunächst auf eine Primzahl p und eine sog. primitive Kongruenzwurzel[4] g mod p.

[4] g ist eine primitive Kongruenzwurzel zur Primzahl p, wenn g^i mod p für $i = \{1, 2, \ldots, p-1\}$ alle Zahlen zwischen 0 und einschließlich $p-1$ erzeugt. Werte für p und g finden sich u.a. in RFC5114.

Wie tauschen Sender und Empfänger die gemeinsamen Schlüssel aus?

Eine Möglichkeit besteht in der Nutzung des Diffie-Helmann-Verfahrens:

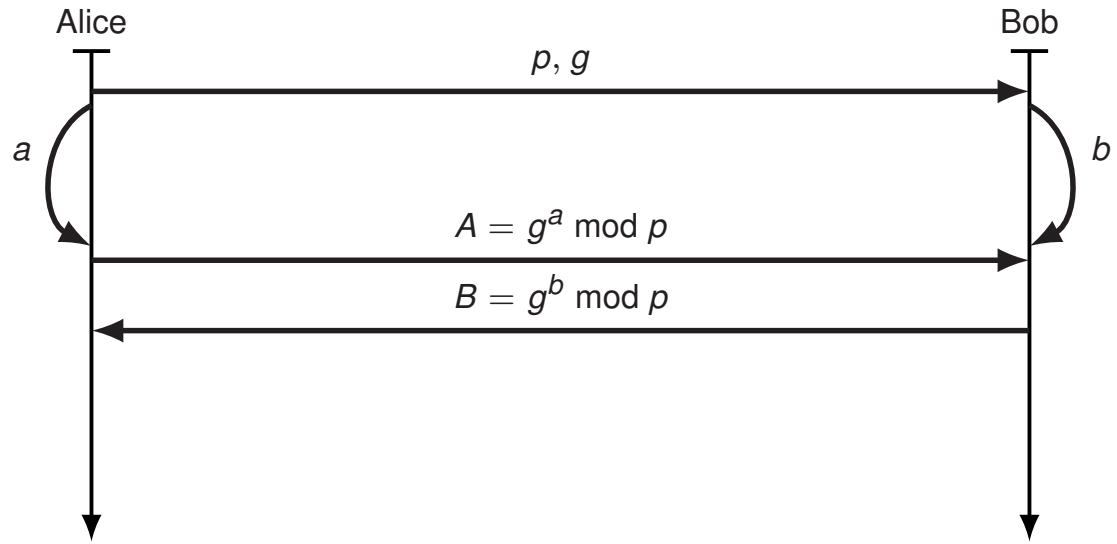


- Alice und Bob einigen sich zunächst auf eine Primzahl p und eine sog. primitive Kongruenzwurzel[4] g mod p.
- Beide erzeugen unabhängig voneinander jeweils eine Zufallszahl a und b.

[4] g ist eine primitive Kongruenzwurzel zur Primzahl p, wenn g^i mod p für $i = \{1, 2, \ldots, p-1\}$ alle Zahlen zwischen 0 und einschließlich $p-1$ erzeugt. Werte für p und g finden sich u.a. in RFC5114.

Wie tauschen Sender und Empfänger die gemeinsamen Schlüssel aus?

Eine Möglichkeit besteht in der Nutzung des Diffie-Helmann-Verfahrens:

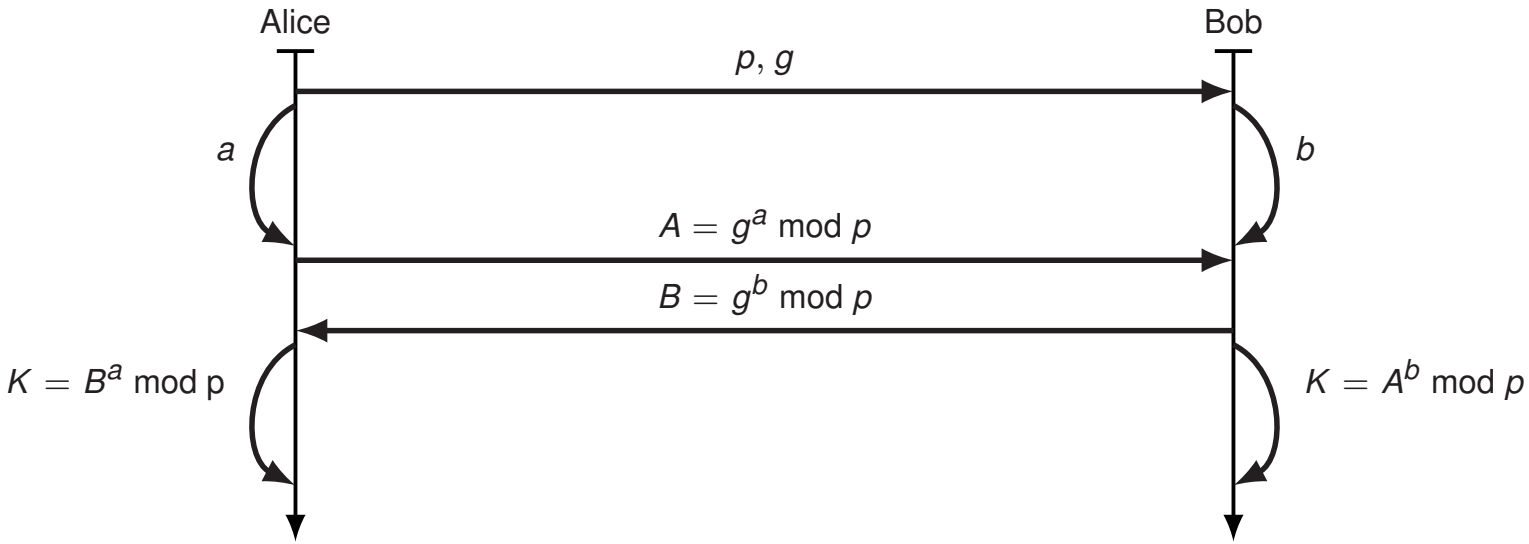


- Alice und Bob einigen sich zunächst auf eine Primzahl p und eine sog. primitive Kongruenzwurzel[4] g mod p.
- Beide erzeugen unabhängig voneinander jeweils eine Zufallszahl a und b.
- Nun werden die Zahlenwerte $B = g^b \bmod p$ bzw. $A = g^a \bmod p$ ausgetauscht. Die Zufallszahlen a und b bleiben aber geheim.

[4] g ist eine primitive Kongruenzwurzel zur Primzahl p, wenn $g^i \bmod p$ für $i = \{1, 2, \ldots, p-1\}$ alle Zahlen zwischen 0 und einschließlich $p-1$ erzeugt. Werte für p und g finden sich u.a. in RFC5114.

Wie tauschen Sender und Empfänger die gemeinsamen Schlüssel aus?

Eine Möglichkeit besteht in der Nutzung des Diffie-Helmann-Verfahrens:



- Alice und Bob einigen sich zunächst auf eine Primzahl p und eine sog. primitive Kongruenzwurzel[4] g mod p.
- Beide erzeugen unabhängig voneinander jeweils eine Zufallszahl a und b.
- Nun werden die Zahlenwerte $B = g^b \bmod p$ bzw. $A = g^a \bmod p$ ausgetauscht. Die Zufallszahlen a und b bleiben aber geheim.
- Alice und Bob können nun jeweils $A^b \bmod p = B^a \bmod p = K$ berechnen.

[4] g ist eine primitive Kongruenzwurzel zur Primzahl p, wenn $g^i \bmod p$ für $i = \{1, 2, \ldots, p-1\}$ alle Zahlen zwischen 0 und einschließlich $p-1$ erzeugt. Werte für p und g finden sich u.a. in RFC5114.

Funktioniert das denn?

Zu zeigen: $K = A^b \bmod p = B^a \bmod p$

$$\begin{aligned} A^b \bmod p &= \left(g^a\right)^b \bmod p \\ &= \left(g^b\right)^a \bmod p \\ &= B^a \bmod p \end{aligned}$$

Ist das Verfahren sicher?

Gegenfrage: Was ist sicher?

- RC4 und das Diffie-Hellmann-Verfahren wie hier vorgestellt sind anfällig gegenüber Man-in-the-Middle-Angriffe:
  - Schafft es Eve, während des Verbindungsaufbaus sowohl die Nachrichten von Alice als auch von Bob abzufangen, kann sie sich für den jeweils anderen Kommunikationspartner ausgeben und unterschiedliche Schlüssel mit Alice und Bob aushandeln.
  - Nach dem Schlüsselaustausch muss Eve weiterhin in der Lage sein, alle Nachrichten von Alice bzw. Bob abzufangen.
  - Diese können dann entschlüsselt, gelesen und modifiziert und anschließend neu verschlüsselt und weitergeleitet werden.
- Aus diesem Grund ist keines der Kriterien Integrität, Authentizität, Vertraulichkeit und Verbindlichkeit wirklich erfüllt.

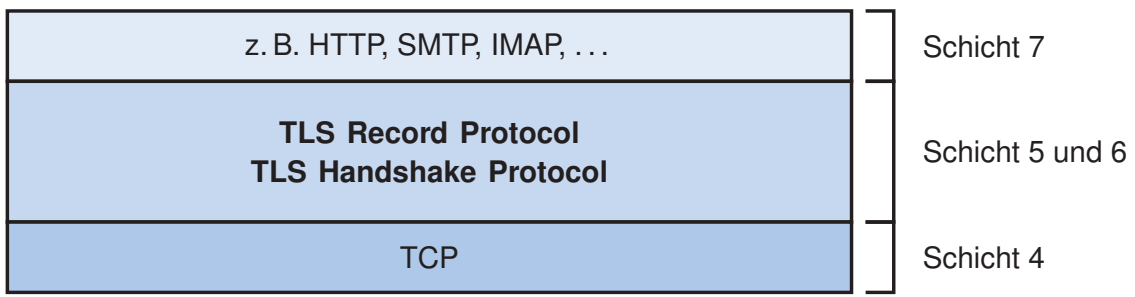
$\Rightarrow$ RC4 und das Diffie-Hellmann-Verfahren alleine sind ein Anfang. Für eine sichere Kommunikation wird aber ein komplettes Protokoll benötigt.

Das Transport Layer Security (TLS) Protokoll

Das Transport Layer Security (TLS) Protokoll ist ein von der IETF standardisiertes Protokoll. In seiner neuesten Version (Version 1.2) ist es durch den RFC 5246 spezifiziert. Ergänzungen dazu finden sich in den RFCs 5746, 5878 und 6176.

TLS setzt auf ein verbindungsorientiertes Transportprotokoll (TCP) auf. Die zu TLS gehörenden Teilprotokolle unterstützen zahlreiche Funktionen. Im Rahmen dieser Vorlesung wird nur ein Teil der Funktionen angesprochen.

- **Handshake Protocol**
  - Aushandlung der Eigenschaften einer Sitzung (Verschlüsselungsalgorithmen, Kompressionsverfahren)
  - Authentifizierung von Instanzen
- **Record Protocol**
  - Ver- und Entschlüsselung
  - Kompression und Dekompression



Dienste von TLS Die in TLS enthaltenen Funktionen dienen zur Realisierung folgender Dienste

- Authentisierung der Instanzen
- Vertraulichkeit der Nachrichten (bzw. der Nutzerdaten)
- Authentisierung der Nachrichten und Integrität der Nachrichten

Diese Dienste werden auf folgende Weise realisiert

- Authentisierung der Instanzen: TLS führt hierzu einen Dialog durch, im Rahmen des TLS Handshake Protocol. Entweder wird nur der Server gegenüber dem Client authentisiert, oder auch (zusätzlich) der Client gegenüber dem Server. Dabei kommen Zertifikate zum Einsatz.
- Vertraulichkeit der Nachrichten: Die ausgetauschten Nutzerdaten werden hierzu mit einem symmetrischen Verschlüsselungsverfahren wie RC4 oder AES verschlüsselt.
- Authentisierung der Nachrichten und Integrität der Nachrichten: Hierzu wird für jede ausgetauschte Nachricht eine kryptografische Hash-Funktion berechnet. In die Berechnung des Hash-Werts fließt ein beiden Seiten bekannter Schlüssel ein.

Zertifikate

- Ein Zertifikat dient dazu, mittels einer kryptografischen Funktion einen öffentlichen Schlüssel einem Identifikator einer Kommunikationsinstanz zuzuordnen.
- Ein Zertifikat wird dadurch erzeugt, dass eine vertrauenswürdige Instanz mit einer elektronischen Signatur bestätigt, dass ein öffentlicher Schlüssel zu einem Identifikator einer Instanz (Name des Rechners oder Adresse des Rechners) gehört.
- Der Standard X.509 legt ein Datenformat für Zertifikate fest. Das Datenformat erlaubt es, unterschiedliche Typen von Identifikatoren, von öffentlichen Schlüssel und von Signaturalgorithmen zu verwenden.
- Bei einer Public Key Infrastruktur (PKI) kommen Zertifikate zum Einsatz, die von sog. Certification Authorities (CAs) ausgestellt werden. Dies sind Organisationen, denen alle Kommunikationspartner vertrauen.

Dienste von TLS

Das Protokoll TLS kann der Sitzungsschicht zugeordnet werden. Das Protokoll erlaubt es den Kommunikationspartnern, einen Sitzungszustand mit gemeinsam bekannten Informationen aufzubauen.

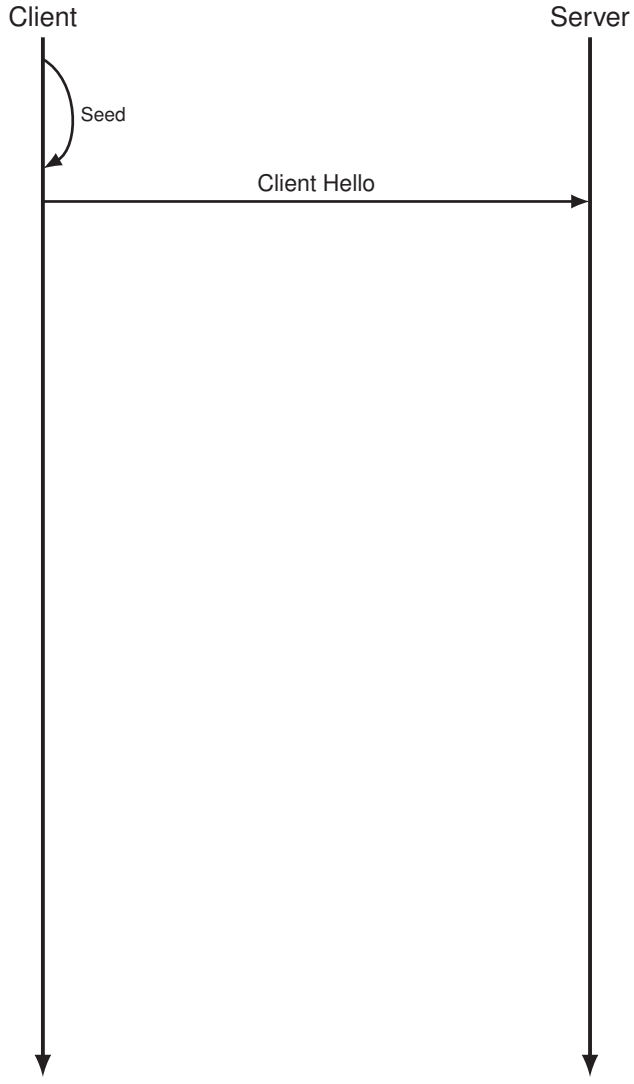
Als Sitzungszustand wird eine Reihe von Informationen (sog. Attribute) verwaltet. Dazu gehören:

- Sitzungsidentifikator (Session ID)
- Zertifikat der Instanz des Kommunikationspartners (Peer Certificate)
- Kryptografische Algorithmen und ihre Parameter (Cypher Suite)
- Funktion zur Kompression (Compression Method)
- Jeweils eine von jeder Instanz generierte Zufallszahl (Seed: ClientHello.random und ServerHello.random)
- Mehrere Schlüssel zur Verschlüsselung und zur Nachrichtenauthentisierung. Zunächst wird ein Premaster-Secret ausgehandelt. Daraus werden dann weitere Schlüssel abgeleitet.

Hinweis: TLS verwendet abhängig von den zur Verfügung stehenden Authentifizierungsmethoden unterschiedliche Handshakes. Im folgenden betrachten wir den TLS Simple Handshake mit serverseitigem Zertifikat.

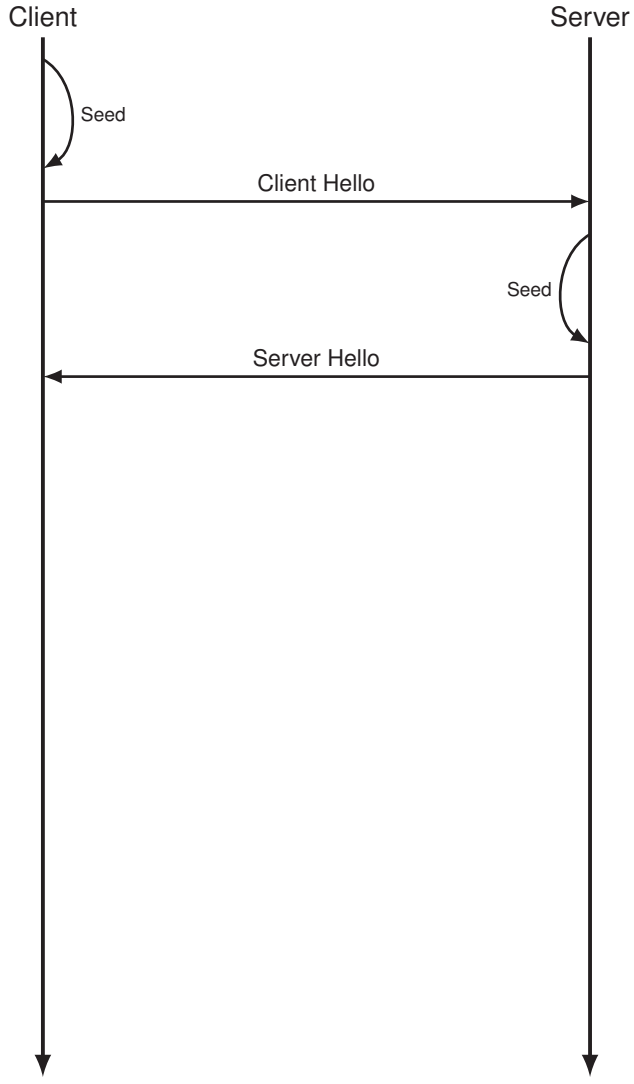
TLS Handshake Protocol

- **Client Hello**
  Unterstützte TLS-Versionen, unterstützte Verschlüsselungsprotokolle und Kompressionsmethoden, Zufallszahl (Seed)



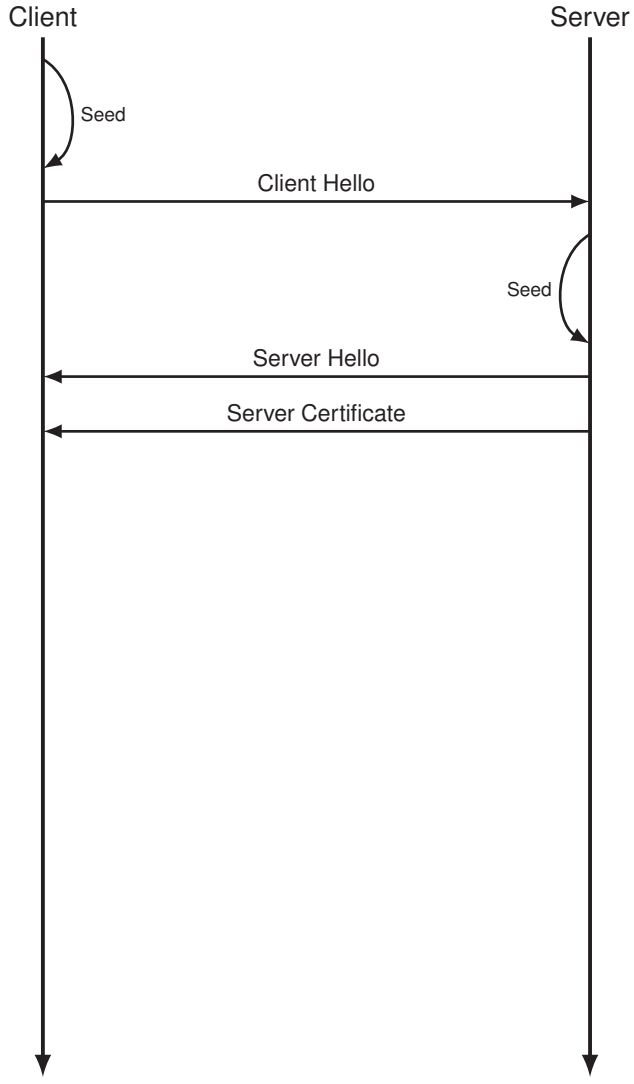
TLS Handshake Protocol

- **Client Hello**
  Unterstützte TLS-Versionen, unterstützte Verschlüsselungsprotokolle und Kompressionsmethoden, Zufallszahl (Seed)
- **Server Hello**
  Gewählte TLS-Version, gewählte Verschlüsselungs- und Kompressionsmethoden, Zufallszahl



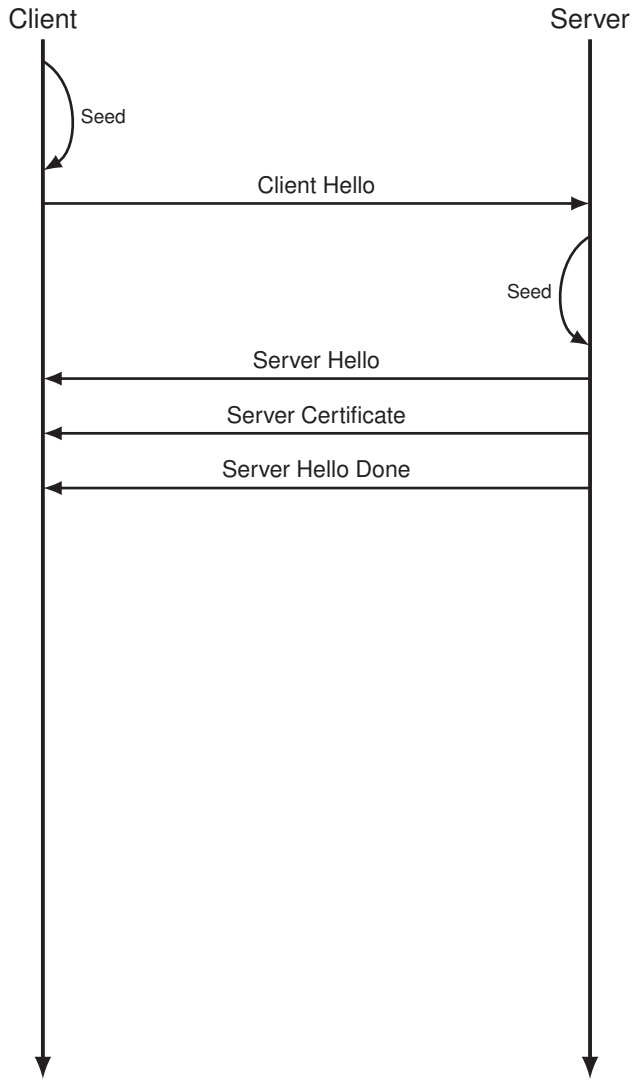
TLS Handshake Protocol

- **Client Hello**
  Unterstützte TLS-Versionen, unterstützte Verschlüsselungsprotokolle und Kompressionsmethoden, Zufallszahl (Seed)
- **Server Hello**
  Gewählte TLS-Version, gewählte Verschlüsselungs- und Kompressionsmethoden, Zufallszahl
- **Server Certificate**
  Enthält den öffentlichen Schlüssel des Servers



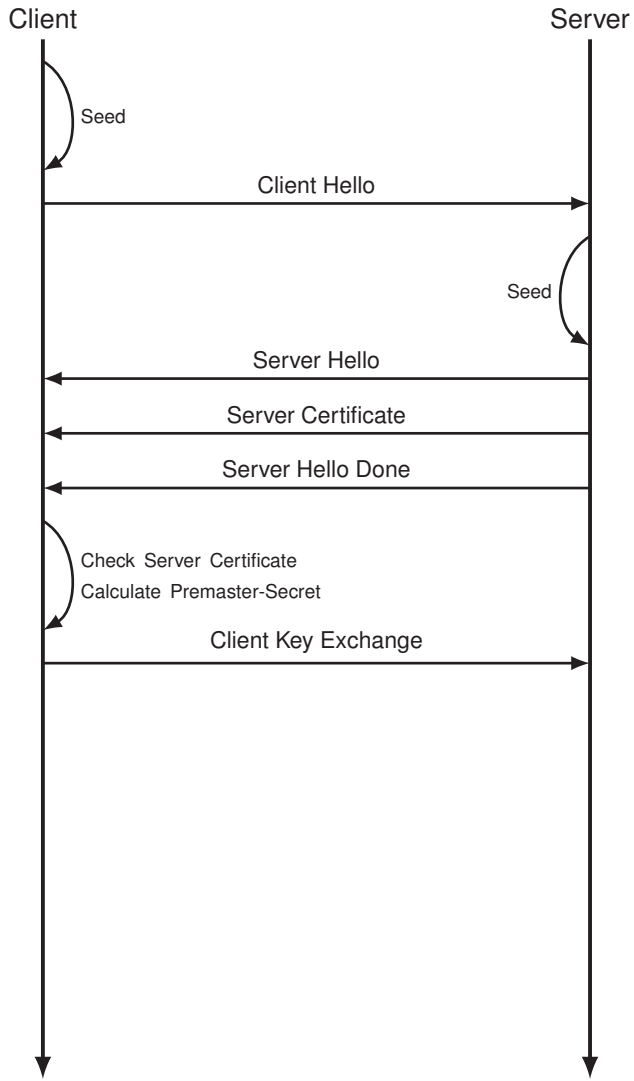
TLS Handshake Protocol

- **Client Hello**
  Unterstützte TLS-Versionen, unterstützte Verschlüsselungsprotokolle und Kompressionsmethoden, Zufallszahl (Seed)
- **Server Hello**
  Gewählte TLS-Version, gewählte Verschlüsselungs- und Kompressionsmethoden, Zufallszahl
- **Server Certificate**
  Enthält den öffentlichen Schlüssel des Servers
- **Server Hello Done**
  Ende des serverseitigen Handshakes



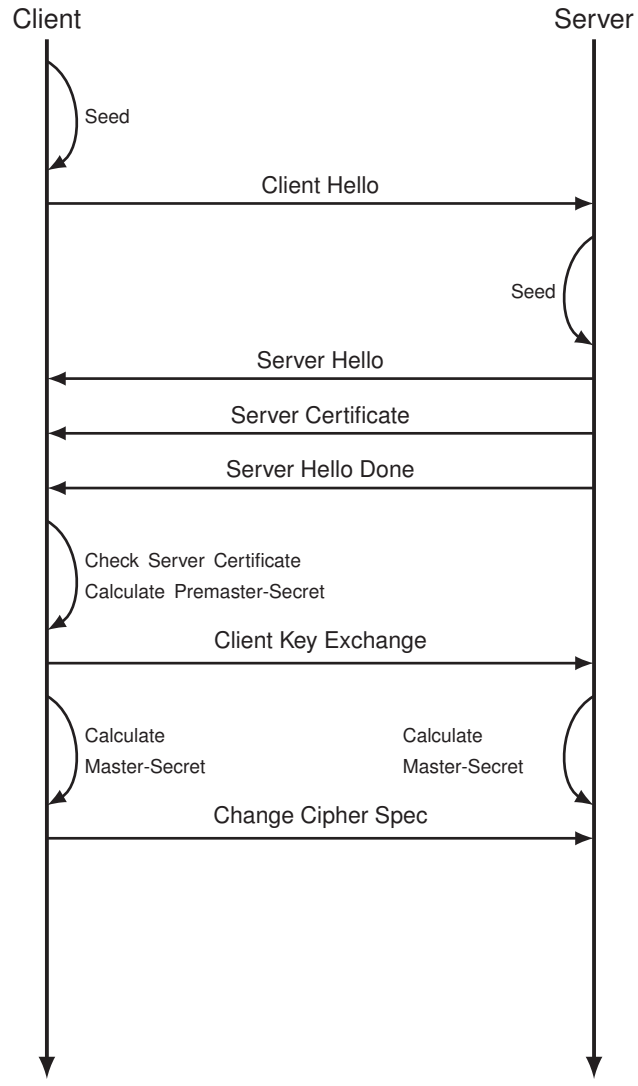
TLS Handshake Protocol

- **Client Hello**
  Unterstützte TLS-Versionen, unterstützte Verschlüsselungsprotokolle und Kompressionsmethoden, Zufallszahl (Seed)
- **Server Hello**
  Gewählte TLS-Version, gewählte Verschlüsselungs- und Kompressionsmethoden, Zufallszahl
- **Server Certificate**
  Enthält den öffentlichen Schlüssel des Servers
- **Server Hello Done**
  Ende des serverseitigen Handshakes
- **Client Key Exchange**
  Mit RSA asymmetrisch verschlüsseltes Premaster-Secret, aus dem zusammen mit den Seeds ein gemeinsames Master-Secret für die symmetrische Verschlüsselung abgeleitet wird



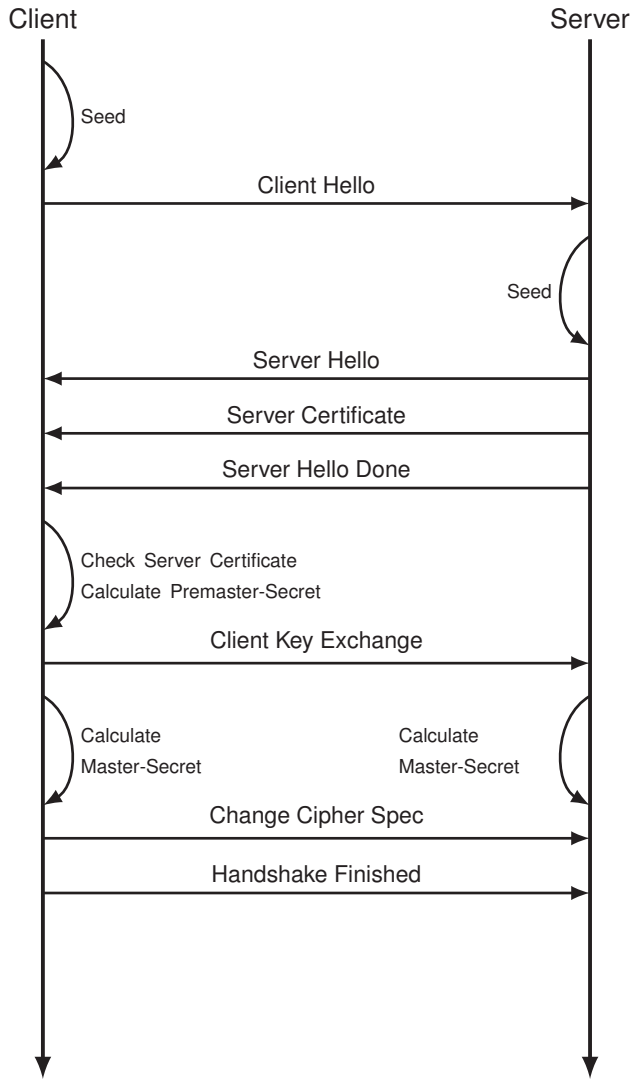
TLS Handshake Protocol

- **Client Hello**
  Unterstützte TLS-Versionen, unterstützte Verschlüsselungsprotokolle und Kompressionsmethoden, Zufallszahl (Seed)
- **Server Hello**
  Gewählte TLS-Version, gewählte Verschlüsselungs- und Kompressionsmethoden, Zufallszahl
- **Server Certificate**
  Enthält den öffentlichen Schlüssel des Servers
- **Server Hello Done**
  Ende des serverseitigen Handshakes
- **Client Key Exchange**
  Mit RSA asymmetrisch verschlüsseltes Premaster-Secret, aus dem zusammen mit den Seeds ein gemeinsames Master-Secret für die symmetrische Verschlüsselung abgeleitet wird
- **Change Cipher Spec**
  Letzte unverschlüsselte Nachricht des Clients



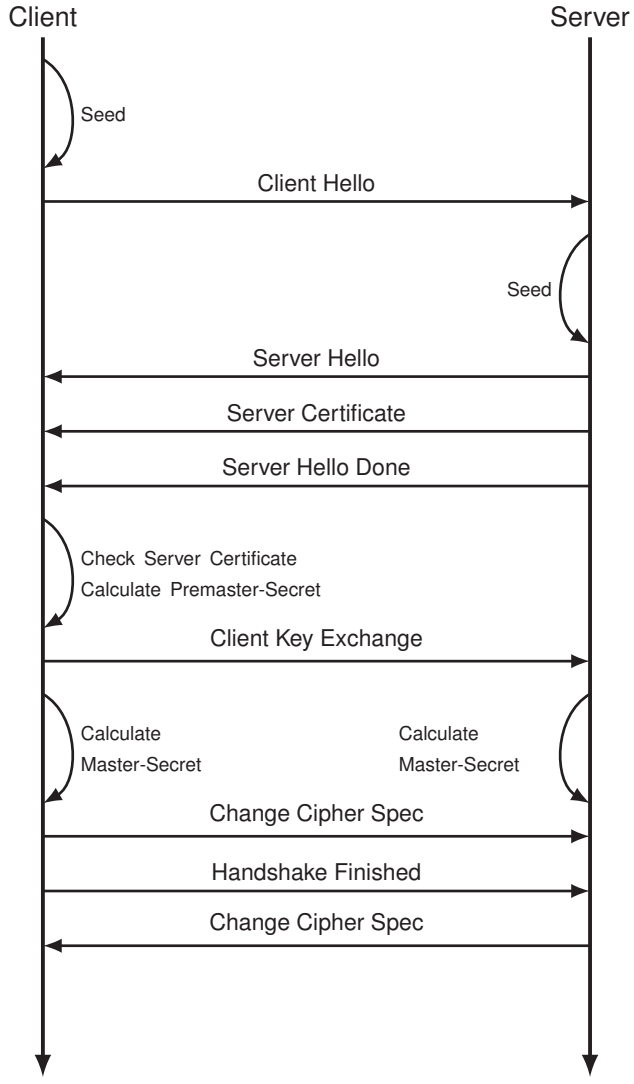
TLS Handshake Protocol

- **Client Hello**
  Unterstützte TLS-Versionen, unterstützte Verschlüsselungsprotokolle und Kompressionsmethoden, Zufallszahl (Seed)
- **Server Hello**
  Gewählte TLS-Version, gewählte Verschlüsselungs- und Kompressionsmethoden, Zufallszahl
- **Server Certificate**
  Enthält den öffentlichen Schlüssel des Servers
- **Server Hello Done**
  Ende des serverseitigen Handshakes
- **Client Key Exchange**
  Mit RSA asymmetrisch verschlüsseltes Premaster-Secret, aus dem zusammen mit den Seeds ein gemeinsames Master-Secret für die symmetrische Verschlüsselung abgeleitet wird
- **Change Cipher Spec**
  Letzte unverschlüsselte Nachricht des Clients
- **Handshake Finished**
  Erste verschlüsselte Nachricht des Clients



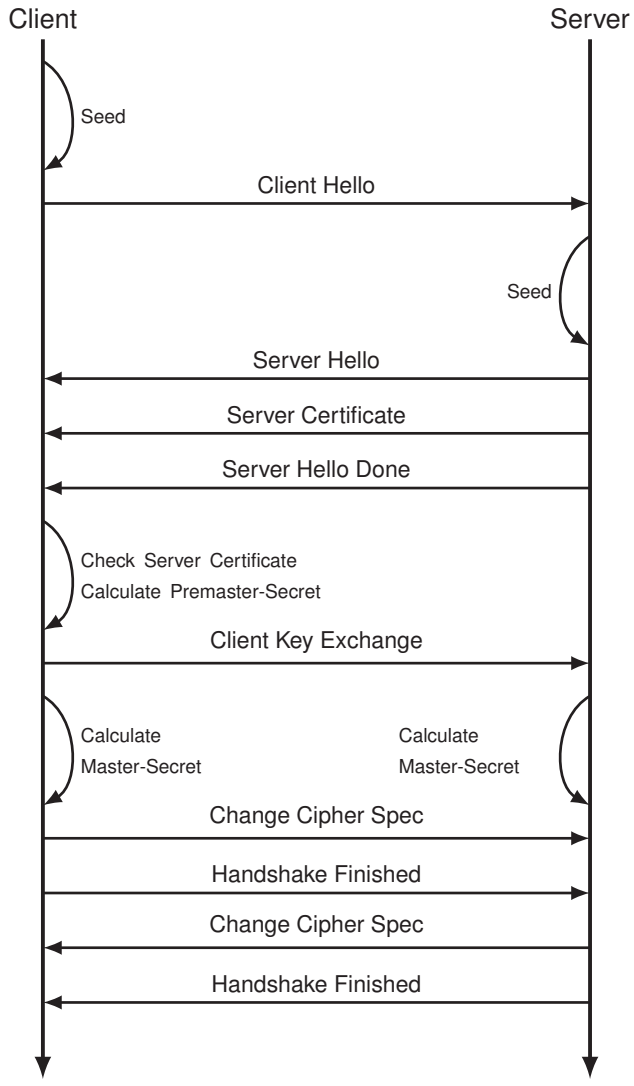
TLS Handshake Protocol

- **Client Hello**
  Unterstützte TLS-Versionen, unterstützte Verschlüsselungsprotokolle und Kompressionsmethoden, Zufallszahl (Seed)
- **Server Hello**
  Gewählte TLS-Version, gewählte Verschlüsselungs- und Kompressionsmethoden, Zufallszahl
- **Server Certificate**
  Enthält den öffentlichen Schlüssel des Servers
- **Server Hello Done**
  Ende des serverseitigen Handshakes
- **Client Key Exchange**
  Mit RSA asymmetrisch verschlüsseltes Premaster-Secret, aus dem zusammen mit den Seeds ein gemeinsames Master-Secret für die symmetrische Verschlüsselung abgeleitet wird
- **Change Cipher Spec**
  Letzte unverschlüsselte Nachricht des Clients
- **Handshake Finished**
  Erste verschlüsselte Nachricht des Clients
- **Change Cipher Spec**
  Letzte unverschlüsselte Nachricht des Servers



TLS Handshake Protocol

- **Client Hello**
  Unterstützte TLS-Versionen, unterstützte Verschlüsselungsprotokolle und Kompressionsmethoden, Zufallszahl (Seed)
- **Server Hello**
  Gewählte TLS-Version, gewählte Verschlüsselungs- und Kompressionsmethoden, Zufallszahl
- **Server Certificate**
  Enthält den öffentlichen Schlüssel des Servers
- **Server Hello Done**
  Ende des serverseitigen Handshakes
- **Client Key Exchange**
  Mit RSA asymmetrisch verschlüsseltes Premaster-Secret, aus dem zusammen mit den Seeds ein gemeinsames Master-Secret für die symmetrische Verschlüsselung abgeleitet wird
- **Change Cipher Spec**
  Letzte unverschlüsselte Nachricht des Clients
- **Handshake Finished**
  Erste verschlüsselte Nachricht des Clients
- **Change Cipher Spec**
  Letzte unverschlüsselte Nachricht des Servers
- **Handshake Finished**
  Erste verschlüsselte Nachricht des Servers



Anmerkungen zum TLS-Handshake

- Der hier gezeigte TLS-Handshake ist der sog. Simple TLS Handshake. Verfügt auch der Client über ein Zertifikat und wird dies vom Server angefordert, so gibt es einen modifzierten Client-Authenticated TLS Handshake.
- Bei Wiederaufnahme einer Session enthalten Client- und Server-Hello eine Session-ID, mit der die zuvor unterbrochene Session identifiziert werden kann. Hierfür gibt es einen abgekürzten Resumed TLS Handshake.
- TLS arbeitet mit unterschiedlichen Anwendungsprotokollen zusammen. Diese werden dann häufig durch ein angehängts „S“ gekennzeichnet, z. B. HTTPS.

Die Themen

- symmetrische Verschlüsselung,
- asymmetrische Verschlüsselung und
- Public Key Infrastructures

werden detaillierter in der Vorlesung Netzsicherheit (Wintersemester) behandelt.

Übersicht

Darstellungsschicht

Die Aufgabe der Darstellungsschicht (engl. Presentation Layer) ist es, den Kommunikationspartnern eine einheitliche Interpretation der Daten zu ermöglichen, d. h. Daten in einem einheitlichen Format zu übertragen.

Der Darstellungsschicht sind grundsätzlich folgende Aufgaben zugeordnet:

- die Darstellung der Daten (Syntax),
- die Datenstrukturen zur Übertragung der Daten
- die Darstellung der Aktionen an diesen Datenstrukturen, sowie
- Datentransformationen.

Die Darstellung auf Schicht 6 muss nicht der Darstellung auf Schicht 7 (Anwendungsschicht) entsprechen. Die Darstellungsschicht ist für die Syntax der Nutzdaten verantwortlich, die Semantik verbleibt bei den Anwendungen.

Unter Syntax versteht man die Darstellung von Daten nach bestimmten Regeln (Grammatik). Werden Daten durch Bedeutung ergänzt, spricht man von Informationen, dies ist die Aufgabe der Semantik.

- Anwendungen sollen syntaxunabhängig miteinander kommunizieren können.
- Anwendungsspezifische Syntax kann von der Darstellungsschicht in eine einheitliche Form umgewandelt und dann übertragen werden.

Aufgaben der Darstellungsschicht

Den grundsätzlichen Aufgaben der Darstellungsschicht lassen sich konkrete Funktionen zuordnen:

- Quellenkodierung und Datenkompression
- Umkodierungen und Übersetzung zwischen Datenformaten
- Strukturierte Darstellung von Informationen
- Ver- und Entschlüsselung

Existierende Protokolle lassen sich nicht immer eindeutig einer Schicht zuordnen. Relevante Protokolle (wie beispielsweise TLS) beinhalten sowohl Funktionen, die üblicherweise der Schicht 5 zugeordnet werden, als auch Funktionen, die üblicherweise der Schicht 6 zugeordnet werden.

Frage: Warum lässt sich das TLS-Protokoll sowohl der Schicht 5 als auch der Schicht 6 zuordnen?

Aufgaben der Darstellungsschicht

Den grundsätzlichen Aufgaben der Darstellungsschicht lassen sich konkrete Funktionen zuordnen:

- Quellenkodierung und Datenkompression
- Umkodierungen und Übersetzung zwischen Datenformaten
- Strukturierte Darstellung von Informationen
- Ver- und Entschlüsselung

Existierende Protokolle lassen sich nicht immer eindeutig einer Schicht zuordnen. Relevante Protokolle (wie beispielsweise TLS) beinhalten sowohl Funktionen, die üblicherweise der Schicht 5 zugeordnet werden, als auch Funktionen, die üblicherweise der Schicht 6 zugeordnet werden.

Frage: Warum lässt sich das TLS-Protokoll sowohl der Schicht 5 als auch der Schicht 6 zuordnen?

- Der TLS Handshake dient dem Aushandeln von Sitzungsschlüssel (engl. Session Keys).
- Dies entspricht dem Aushandeln der Kommunikationsparameter einer Sitzung und kann somit der Sitzungsschicht zugeordnet werden.
- Bei den Funktionen Kompression sowie Ver- und Entschlüsseln von Nutzdaten handelt es sich um Funktionen, die der Darstellungsschicht zugeordnet werden.
- TLS lässt sich somit sowohl der Schicht 5 als auch der Schicht 6 zuordnen.

Datenkompression und Umkodierung

Datenkompression und Umkodierung gehört zu den wichtigsten Funktionen der Darstellungsschicht. Es ist wichtig die zugrundeliegenden Prozessen zu unterscheiden.

Definition (Umkodierung)

Umkodierung oder Transkodierung von Daten beschreibt den Prozess der Überführung von einer Darstellung in eine andere. Hierbei sind keine Einschränkungen wie bezüglich der Anzahl verwendeter Zeichen o. ä. nötig. Eine gültige Umkodierung kann gegebenen Nutzdaten in ihrem Umfang auch erweitern.

Definition (Kompression)

Unter Datenkompression versteht man die Entfernung von Redundanz. Die komprimierte Nachricht ist i. A. kürzer als das Original.

- Wir wollen im Folgenden näher auf Kompression eingehen.
- Als Beispiel für ein Kompressionsverfahren werden wir den Huffman-Algorithmus betrachten.
- Zudem wollen wir wichtige Eigenschaften des Huffman-Algorithmus näher analysieren.

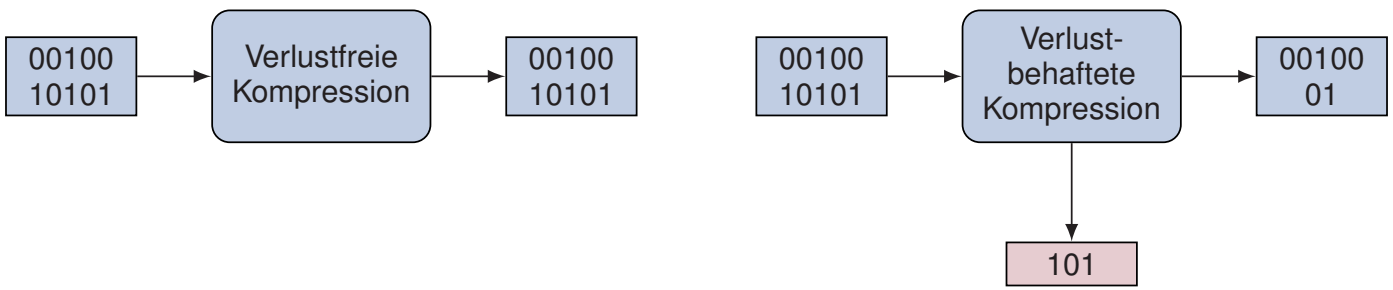
Einteilung von Datenkompression und Umkodierung

Kodierungsverfahren können unterschieden werden nach

- Fixed-Length Code bei welchen alle Zeichen mit der gleichen Anzahl Bits kodiert werden, z.B. ASCII (8 Bits) oder UNICODE (16 Bits).
- Variable-Length Code bei welchen alle Zeichen abhängig von ihrer Auftrittswahrscheinlichkeit mit einer Kodierungslänge korreliert werden.

Es lassen sich zwei grundlegende Arten der Kompression unterscheiden:

- Verlustfreie Komprimierung (engl. Lossless Data Compression): Die dekodierten Daten können identisch zum Original wieder hergestellt werden, ohne das Informationen verloren gehen oder verändert werden (wie z.B. beim ZIP-Dateiformat, das den Einsatz unterschiedlicher verlustfreier Kompressionsalgorithmen ermöglicht).
- Verlustbehaftete Komprimierung (engl. Lossy Data Compression): Bei der Dekodierung gehen Informationen der ursprünglichen Daten verloren und können nicht mehr vollständig richtig dekodiert werden (siehe z.B. das verlustbehaftete Kompressionsverfahren JPEG).



Beispiel: Kompression mittels Huffman-Codes

- Viele Protokolle komprimieren Daten vor dem Senden (Quellenkodierung).
- TLS beispielsweise bietet optional Kompressionsmethoden. Diese werden vor der Verschlüsselung angewandt. (Warum davor?)
- Ein häufig (u. a. von TLS) verwendetes Kompressionsverfahren für Texte ist der Huffman-Code.

Grundlegende Idee der Huffman-Kodierung:

- Nicht alle Textzeichen treten mit derselben Häufigkeit auf, z. B. tritt der Buchstabe „E“ in der deutschen Sprache mit einer Häufigkeit von 17.4 % gefolgt von „N“ mit 9.78 % auf.
- Anstelle Zeichen mit uniformer Codewortlänge zu kodieren, (z. B. ASCII-Code), weise häufigen Zeichen kürzere Codewörter zu.

Bei der Huffman-Kodierung handelt es sich also um ein

- verlustfreies Kompressionsverfahren.

Konstruktion eines Huffman-Codes:

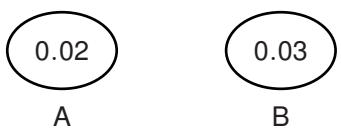
Gegeben Sei das Alphabet $\mathcal{A} = \{A, B, C, D, E, F, G, H\}$ sowie Auftrittswahrscheinlichkeiten $\Pr[X = z]$ für alle Zeichen $z \in \mathcal{A}$. Es wird außerdem vorausgesetzt, dass die einzelnen Zeichen unabhängig voneinander auftreten.

z	$\Pr[X = z]$
A	0.02
B	0.03
C	0.05
D	0.08
E	0.12
F	0.15
G	0.25
H	0.30

Konstruktion eines Huffman-Codes:

Gegeben Sei das Alphabet $\mathcal{A} = \{A, B, C, D, E, F, G, H\}$ sowie Auftrittswahrscheinlichkeiten $\Pr[X = z]$ für alle Zeichen $z \in \mathcal{A}$. Es wird außerdem vorausgesetzt, dass die einzelnen Zeichen unabhängig voneinander auftreten.

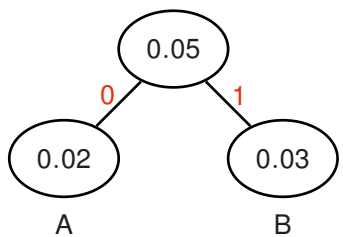
z	$\Pr[X = z]$
A	0.02
B	0.03
C	0.05
D	0.08
E	0.12
F	0.15
G	0.25
H	0.30



Konstruktion eines Huffman-Codes:

Gegeben Sei das Alphabet $\mathcal{A} = \{A, B, C, D, E, F, G, H\}$ sowie Auftrittswahrscheinlichkeiten $\Pr[X = z]$ für alle Zeichen $z \in \mathcal{A}$. Es wird außerdem vorausgesetzt, dass die einzelnen Zeichen unabhängig voneinander auftreten.

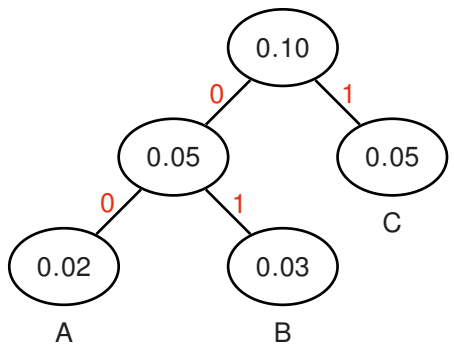
z	$\Pr[X = z]$
A	0.02
B	0.03
C	0.05
D	0.08
E	0.12
F	0.15
G	0.25
H	0.30



Konstruktion eines Huffman-Codes:

Gegeben Sei das Alphabet $\mathcal{A} = \{A, B, C, D, E, F, G, H\}$ sowie Auftrittswahrscheinlichkeiten $\Pr[X = z]$ für alle Zeichen $z \in \mathcal{A}$. Es wird außerdem vorausgesetzt, dass die einzelnen Zeichen unabhängig voneinander auftreten.

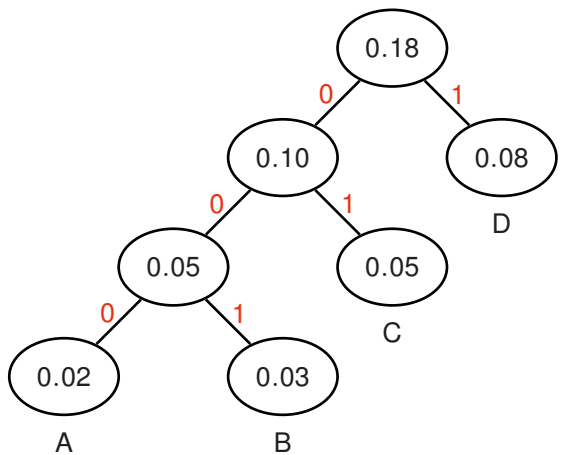
z	$\Pr[X = z]$
A	0.02
B	0.03
C	0.05
D	0.08
E	0.12
F	0.15
G	0.25
H	0.30



Konstruktion eines Huffman-Codes:

Gegeben Sei das Alphabet $\mathcal{A} = \{A, B, C, D, E, F, G, H\}$ sowie Auftrittswahrscheinlichkeiten $\Pr[X = z]$ für alle Zeichen $z \in \mathcal{A}$. Es wird außerdem vorausgesetzt, dass die einzelnen Zeichen unabhängig voneinander auftreten.

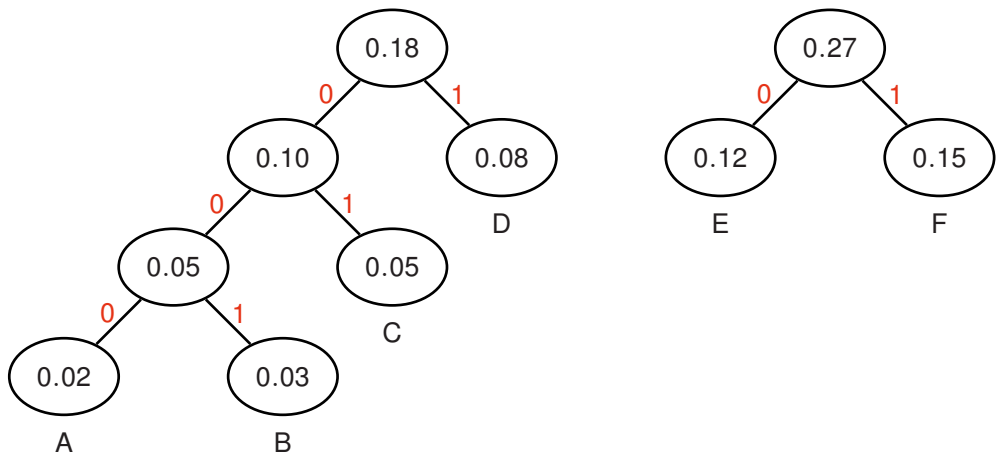
z	$\Pr[X = z]$
A	0.02
B	0.03
C	0.05
D	0.08
E	0.12
F	0.15
G	0.25
H	0.30



Konstruktion eines Huffman-Codes:

Gegeben Sei das Alphabet $\mathcal{A} = \{A, B, C, D, E, F, G, H\}$ sowie Auftrittswahrscheinlichkeiten $\Pr[X = z]$ für alle Zeichen $z \in \mathcal{A}$. Es wird außerdem vorausgesetzt, dass die einzelnen Zeichen unabhängig voneinander auftreten.

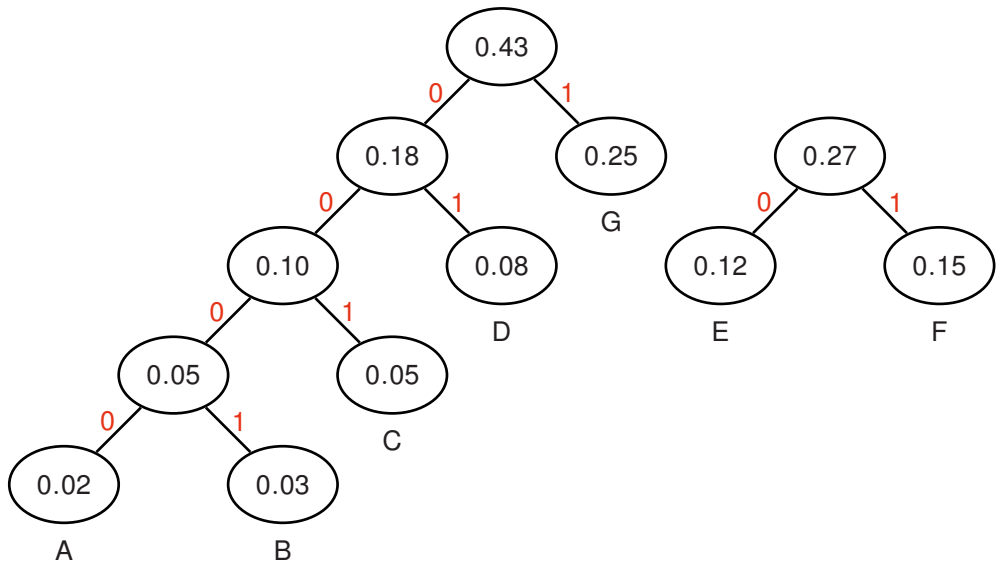
z	$\Pr[X = z]$
A	0.02
B	0.03
C	0.05
D	0.08
E	0.12
F	0.15
G	0.25
H	0.30



Konstruktion eines Huffman-Codes:

Gegeben Sei das Alphabet $\mathcal{A} = \{A, B, C, D, E, F, G, H\}$ sowie Auftrittswahrscheinlichkeiten $\Pr[X = z]$ für alle Zeichen $z \in \mathcal{A}$. Es wird außerdem vorausgesetzt, dass die einzelnen Zeichen unabhängig voneinander auftreten.

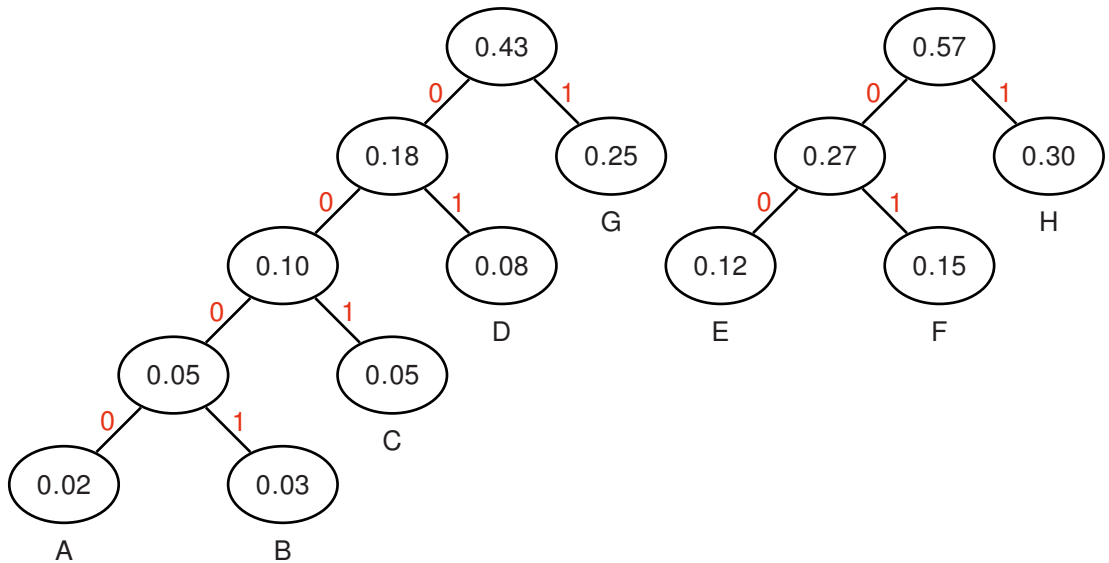
z	$\Pr[X = z]$
A	0.02
B	0.03
C	0.05
D	0.08
E	0.12
F	0.15
G	0.25
H	0.30



Konstruktion eines Huffman-Codes:

Gegeben Sei das Alphabet $\mathcal{A} = \{A, B, C, D, E, F, G, H\}$ sowie Auftrittswahrscheinlichkeiten $\Pr[X = z]$ für alle Zeichen $z \in \mathcal{A}$. Es wird außerdem vorausgesetzt, dass die einzelnen Zeichen unabhängig voneinander auftreten.

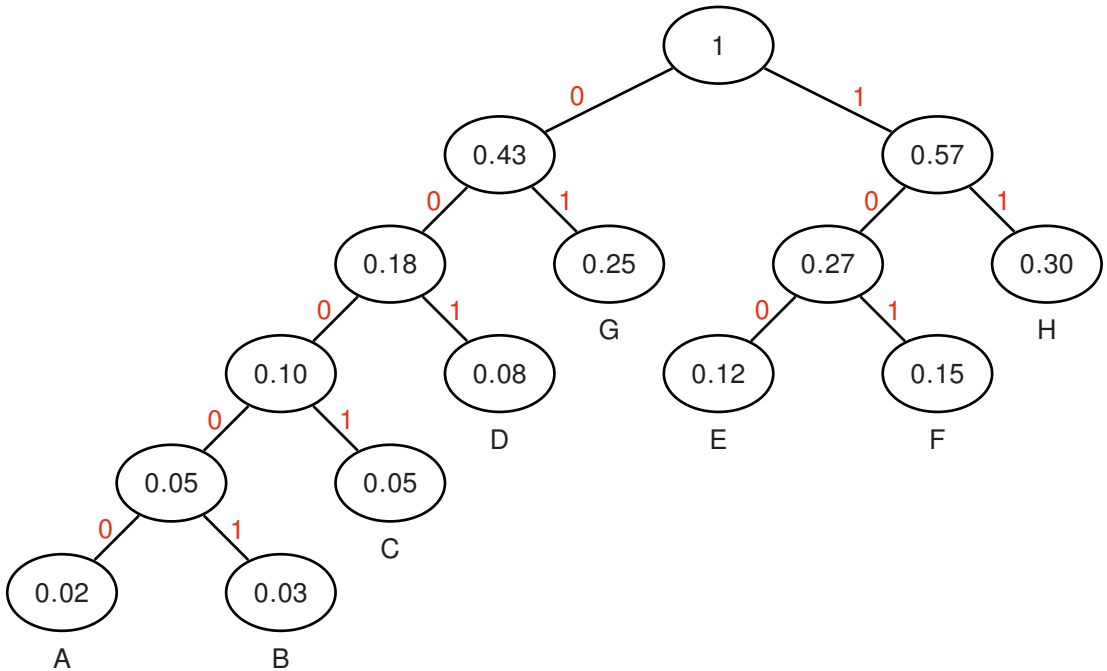
z	$\Pr[X = z]$
A	0.02
B	0.03
C	0.05
D	0.08
E	0.12
F	0.15
G	0.25
H	0.30



Konstruktion eines Huffman-Codes:

Gegeben Sei das Alphabet $\mathcal{A} = \{A, B, C, D, E, F, G, H\}$ sowie Auftrittswahrscheinlichkeiten $\Pr[X = z]$ für alle Zeichen $z \in \mathcal{A}$. Es wird außerdem vorausgesetzt, dass die einzelnen Zeichen unabhängig voneinander auftreten.

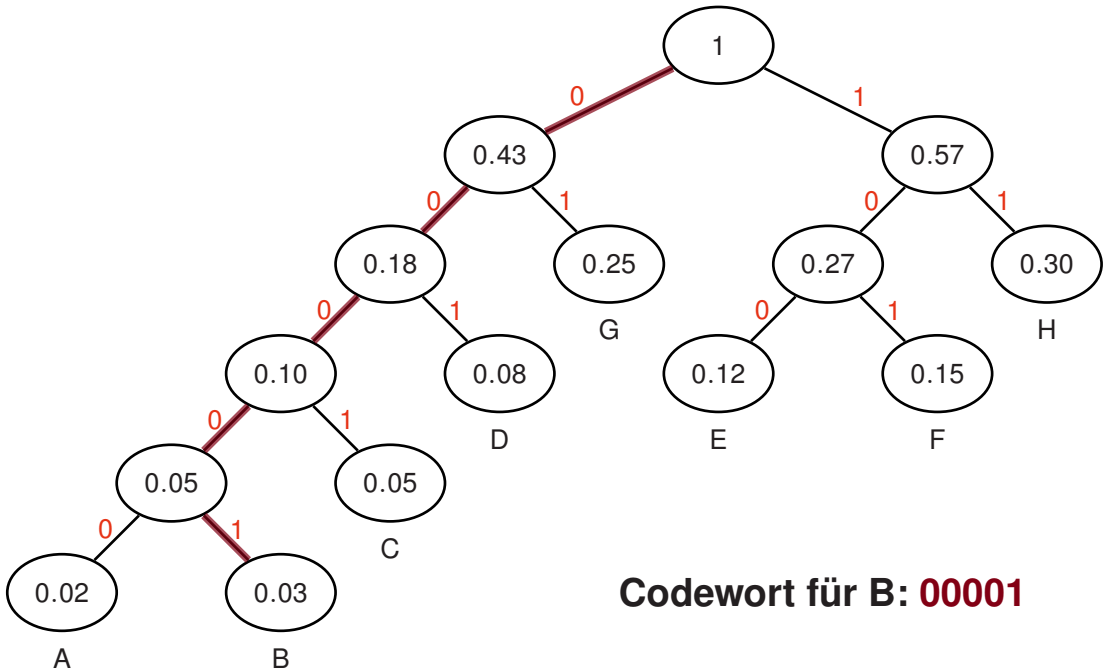
z	$\Pr[X = z]$
A	0.02
B	0.03
C	0.05
D	0.08
E	0.12
F	0.15
G	0.25
H	0.30



Konstruktion eines Huffman-Codes:

Gegeben Sei das Alphabet $\mathcal{A} = \{A, B, C, D, E, F, G, H\}$ sowie Auftrittswahrscheinlichkeiten $\Pr[X = z]$ für alle Zeichen $z \in \mathcal{A}$. Es wird außerdem vorausgesetzt, dass die einzelnen Zeichen unabhängig voneinander auftreten.

z	$\Pr[X = z]$
A	0.02
B	0.03
C	0.05
D	0.08
E	0.12
F	0.15
G	0.25
H	0.30



Codewort für B: 00001

Durchschnittliche Codewortlänge

z	$\Pr[X = z]$	Huffman-Code	Länge $l_H(z)$	„Einfacher“ Code
A	0.02	00000	5	000
B	0.03	00001	5	001
C	0.05	0001	4	010
D	0.08	001	3	011
E	0.12	010	3	100
F	0.15	011	3	101
G	0.25	10	2	110
H	0.30	11	2	111

- Uniformer Code:

$$\mathrm{E}[l(z)] = 3.0, \quad \text{da alle Codewörter gleich lang sind}$$

- Huffman-Code:

$$\mathrm{E}[l_H(z)] = \sum_{z \in \mathcal{A}} \Pr[X = z] \cdot l_H(z) = 2.6$$

$$\Rightarrow \text{Die Einsparung beträgt } 1 - \mathrm{E}[l_H(z)]/\mathrm{E}[l(z)] \approx 13\,\%$$

Anmerkungen zum Huffman-Code:

- Statische Huffman-Codes sind darauf angewiesen, dass die Auftrittswahrscheinlichkeit der Zeichen den Erwartungen entspricht.
- Zeichenhäufigkeiten können dynamisch bestimmt werden, allerdings muss dem Empfänger dann das verwendete Codebuch mitgeteilt werden.
- Längere Codewörter (z. B. ganze Wörter statt einzelner Zeichen) werden infolge der Komplexität zum Bestimmen des Codebuchs ein Problem.
- Der Huffman-Code ist ein optimaler und präfixfreier Code.

Definition (Optimaler Präfixcode)

Bei einem Präfixfreien Code sind gültige Codewörter niemals Präfix eines anderen Codeworts desselben Codes. Ein optimaler Präfixfreier Code minimiert darüber hinaus die mittlere Codewortlänge

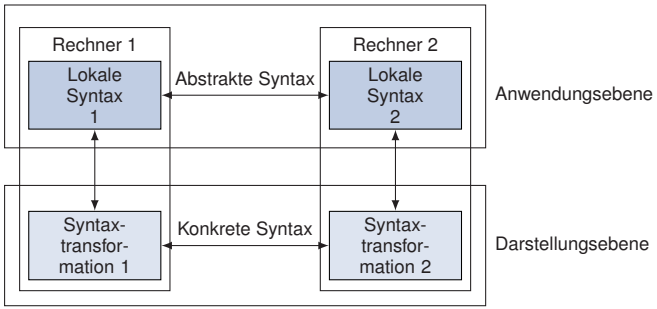
$$\sum_{i \in \mathcal{A}} p(i) \cdot |c(i)|,$$

wobei $p(i)$ die Auftrittswahrscheinlichkeit von $i \in \mathcal{A}$ und $c(i)$ die Abbildung auf ein entsprechendes Codewort bezeichnen.

- Es handelt sich zudem um einen Variable-Length Code.
- Die Huffman-Codierung zählt zu den sogenannten Entropy-Encoding Verfahren, welche im Anschluss näher analysiert werden sollen.

Einheitliche Syntax

Der ITU Standard X.208 schlägt die folgende Methodik vor, um Daten mit unterschiedlichen syntaktischen Gegebenheiten zwischen zwei Systemen zu übertragen.



- Abstrakte Syntax beschreibt die Menge aller Darstellungstypen, die durch die Anwendungsprozesse definiert wurden.
- Die Kodierung dieser Darstellungstypen im lokalen System wird als lokale Syntax bezeichnet.
- Die abstrakte Syntax wird mittels einer sog. Transfersyntax zwischen den Instanzen über die Darstellungsschicht transferiert.
- Die Abbildung von einer abstrakten Syntax in eine konkrete Transfersyntax wird mittels Kodierregeln (engl. Encoding Rules) erreicht.
- Als Presentation Context bestimmen die Übergangsregeln für die De- und Enkodierung zwischen abstrakter Syntax und Transfersyntax.

Abstrakte Syntaxnotation Nummer 1: ASN.1

- ASN.1 ist eine abstrakte Syntax.
- Sie ist in der Backus-Naur-Notation gegeben und wird als Semantiksprache verwendet.

Definition (Backus-Naur-Form)

Die Backus-Naur-Form stellt eine Möglichkeit dar, die Syntax einer formalen Sprache zu beschreiben. In den BNF-Regeln sind folgende Elemente zugelassen:

- Syntaktische Variablen bzw. Nichtterminalsymbole (engl. nonterminals) dienen als Platzhalter für syntaktische Elemente. Nichtterminalsymbole werden mit <> beschrieben.
- Das Symbol **::=** dient einer Zuweisung.
- Terminalsymbole werden einzelnen Zeichen oder Zeichenketten der Wörter der Sprache bezeichnet.
- Operatoren ermöglichen die Verknüpfung von Terminalen und/oder Nichtterminalen.

Für die Verknüpfungen stehen folgenden Möglichkeiten zur Verfügung:

- Verkettung / Konkatenation
- Klammerung mehrere Ausdrücke: ( )
- Auswahl verschiedener Ausdrücke: |
- Wiederholungen einzelner Zeichen: *, um mehrere Zeichen zu wiederholen werden geschweifte Klammern { } verwendet
- optionale Ausdrücke: []

ASN.1 Beispiel

- Bei ASN.1 werden zusammengehörende Definitionen in Modulen gegliedert.
- Module können Definitionen exportieren (declarations) oder importieren (linkage)

```
ModulName DEFINITIONS ::= BEGIN
    linkage
    declarations
END
```

Definierbare Objekte in ASN.1 umfassen folgende Elemente:

- Typen (Types), um neue Datenstrukturen zu definieren, und werden mit Großbuchstaben am Beginn gekennzeichnet (`DatenTyp`)
- Werte (Values), um die Ausprägung der Typen zu definieren, werden lediglich in Kleinbuchstaben geschrieben (`zustand`)
- Makros (Macros), um die Grammatik der Sprache ändern zu können, werden wie alle anderen Schlüsselwörter in ASN.1 mit Großbuchstaben versehen (`OBJEKT`).

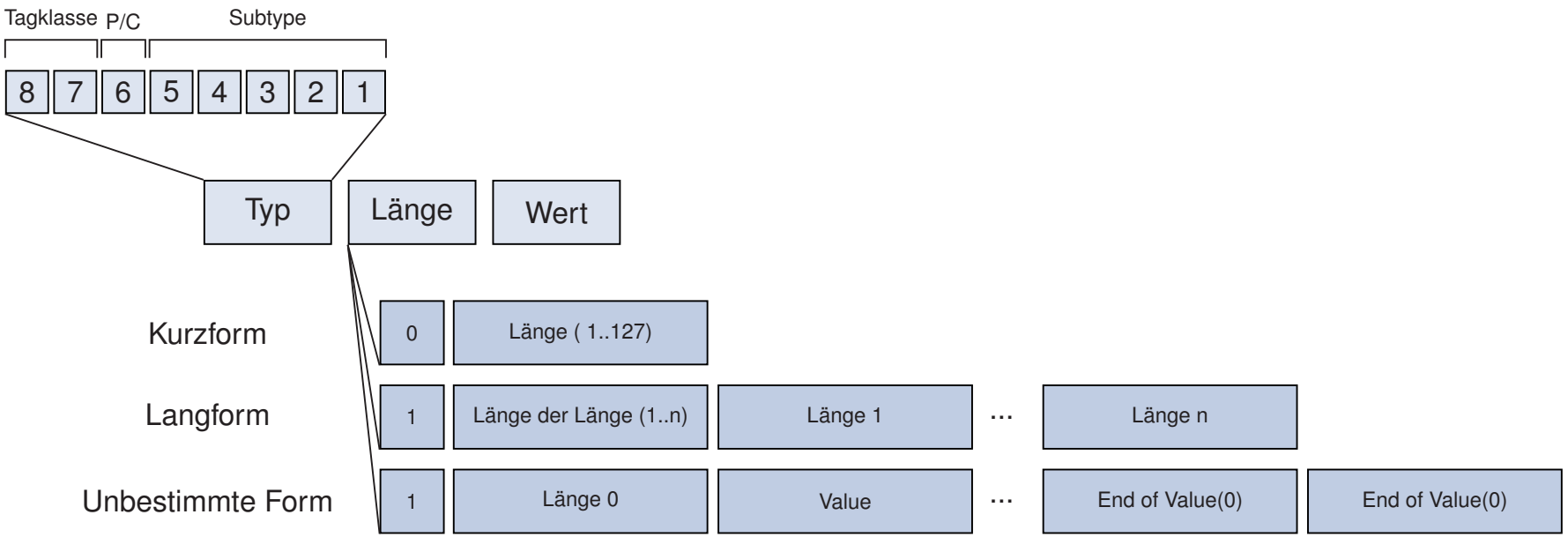
- Ein Typ wir durch `TypName ::= WERT` eingeführt, Variablen werden durch `VariablenName TypName ::= WERT` beschrieben.
- ASN.1 stellt eine Vielzahl von Datentypen bereit, die hier nicht näher beschrieben werden sollen.
- Eine genaue Beschreibung der Sprache findet sich in X.208 bzw. IS 8824 (für interessierte Studenten).

Basic Encoding Rules (BER)

Definition (Basic Encoding Rules, BER)

Die Basic Encoding Rules (BER) legen eine konkrete Transfersyntax fest, die definiert, wie die Datentypen der ASN.1 bei der Übertragung dargestellt werden. Sie sind in ISO 8825 definiert. Die Reihenfolge der Bits in einem Oktett und die Ordnung der Bytes selbst beginnt mit den Most Significant Bit (MSB) am Anfang.

In BER werden die Datentypen nach ASN.1 wie folgt kodiert und werden kurz mit TLV beschrieben:



Anmerkungen zur Kodierung in BER: Typ- oder Tag-Feld

Das Typ- bzw. Tag Feld enthält Informationen über die Klasse in den Bits 8 und 7, welches als class-Element bezeichnet wird. Mögliche Kodierungen sind

- Universal (**00**): ist reserviert für die Typen im ASN.1 Standard
- Application (**01**): ist gültig für eine Anwendung
- Context-Specific (**10**): ist gültig für eigene Spezifikationen
- Private (**11**): unterliegt keinen Einschränkungen, die Bedeutung geht aus dem Kontext hervor

Bit Nummer 6 wird als P/C-Bit bezeichnet und gibt an, ob es sich um einen primitiven oder konstrurierten Typ handelt.

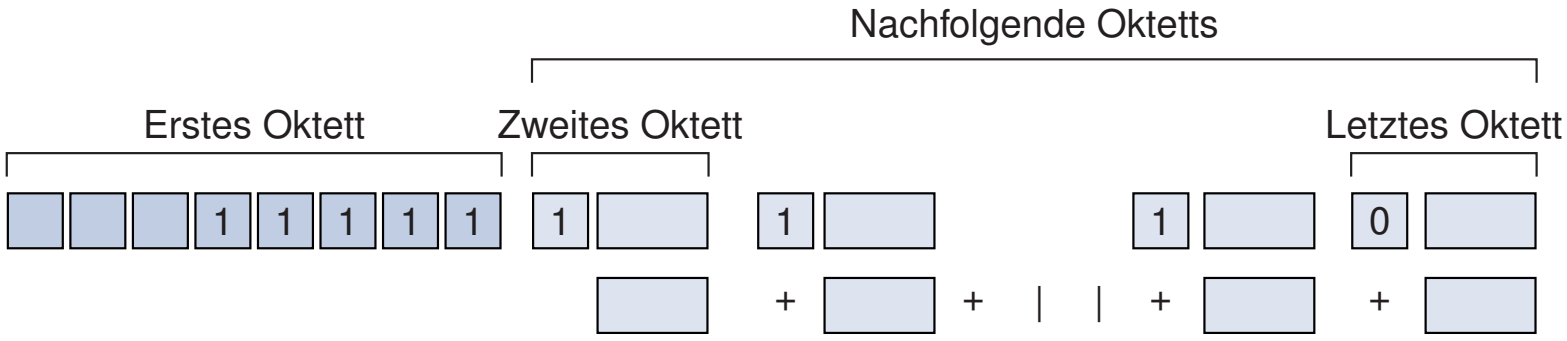
- **P/C = 1**: Der Value-Teil enthält wiederum eine BER-Struktur (TLV) enthält.
- **P/C = 0**: Der Value-Teil enthält nur ein einfaches Element

In den Bits 5 bis 1 wir der Subtype des Elements kodiert. Für die Klasse UNIVERSAL kann beispielsweise folgendes verwendet werden:

- **0**: End of Content (EOC)
- **1**: Boolean
- **2**: Integer
- Werte bis einschließlich 30 identifizieren weitere Subtypen, z.B. 4: OCTET STRING, 5: NULL
- **31**: Continued Number type: Die Identifikation des Subtypes befindet sich in einem späteren Feld

Anmerkungen zur Kodierung in BER: Continued Number Type

Der Continued Number Type hat folgende Struktur:



- Bit 8 jedes Oktetts wird auf 1 gesetzt um anzuzeigen, dass weitere Subtypefelder-Felder folgen.
- Wird das 8. Bit des Oktetts auf 0 gesetzt, ist das Ende des Subtypes erreicht.
- Diese Darstellung wird verwendet, wenn der Subtype durch einen größeren Wert als 30 identifiziert wird.

Anmerkungen zur Kodierung in BER: Längenfeld

Das Längenfeld gibt an, wie viele Datenoktetts folgen. Es können verschiedene Formen unterschieden werden:

- Kurzform: Die Werte liegen zwischen 0 und 127.
- Langform: Das MSB (Most Significant Bit) wird auf 1 gesetzt und die Bits 7 bis 1 kodieren die Anzahl nachfolgender Oktette (127 ist reserviert für eventuelle Erweiterungen).
- Unbestimmte Form: Es werden solange Oktetts übertragen bis der Delimiter (zwei auf Null gesetzte Oktette) verschickt werden. Es ist darauf zu achten, dass diese Struktur nicht in den zuversendenden Daten erscheint (s. Character Stuffing, Kapitel 2).

Anmerkungen zur Kodierung in BER: Längenfeld

Das Längenfeld gibt an, wie viele Datenoktetts folgen. Es können verschiedene Formen unterschieden werden:

- Kurzform: Die Werte liegen zwischen 0 und 127.
- Langform: Das MSB (Most Significant Bit) wird auf 1 gesetzt und die Bits 7 bis 1 kodieren die Anzahl nachfolgender Oktette (127 ist reserviert für eventuelle Erweiterungen).
- Unbestimmte Form: Es werden solange Oktetts übertragen bis der Delimiter (zwei auf Null gesetzte Oktette) verschickt werden. Es ist darauf zu achten, dass diese Struktur nicht in den zuversendenden Daten erscheint (s. Character Stuffing, Kapitel 2).

Frage: Wieviele Byte können mit Hilfe der jeweiligen Längenfelder übertragen werden?

Anmerkungen zur Kodierung in BER: Längenfeld

Das Längenfeld gibt an, wie viele Datenoktetts folgen. Es können verschiedene Formen unterschieden werden:

- Kurzform: Die Werte liegen zwischen 0 und 127.
- Langform: Das MSB (Most Significant Bit) wird auf 1 gesetzt und die Bits 7 bis 1 kodieren die Anzahl nachfolgender Oktette (127 ist reserviert für eventuelle Erweiterungen).
- Unbestimmte Form: Es werden solange Oktetts übertragen bis der Delimiter (zwei auf Null gesetzte Oktette) verschickt werden. Es ist darauf zu achten, dass diese Struktur nicht in den zuversendenden Daten erscheint (s. Character Stuffing, Kapitel 2).

Frage: Wieviele Byte können mit Hilfe der jeweiligen Längenfelder übertragen werden?

- Mit Hilfe der Kurzform: maximal 127 Byte. Die Langform erlaubt $126 * 8 = 1008$ Bit zur Kodierung der Länge: $2,47 * 10^{303}$ Byte

Anmerkungen zur Kodierung in BER: Längenfeld

Das Längenfeld gibt an, wie viele Datenoktetts folgen. Es können verschiedene Formen unterschieden werden:

- Kurzform: Die Werte liegen zwischen 0 und 127.
- Langform: Das MSB (Most Significant Bit) wird auf 1 gesetzt und die Bits 7 bis 1 kodieren die Anzahl nachfolgender Oktette (127 ist reserviert für eventuelle Erweiterungen).
- Unbestimmte Form: Es werden solange Oktetts übertragen bis der Delimiter (zwei auf Null gesetzte Oktette) verschickt werden. Es ist darauf zu achten, dass diese Struktur nicht in den zuversendenden Daten erscheint (s. Character Stuffing, Kapitel 2).

Frage: Wieviele Byte können mit Hilfe der jeweiligen Längenfelder übertragen werden?

- Mit Hilfe der Kurzform: maximal 127 Byte. Die Langform erlaubt $126 * 8 = 1008$ Bit zur Kodierung der Länge: $2,47 * 10^{303}$ Byte

Frage: Welchen Vorteil bringt eine unbestimmte Form, wenn die Anzahl der zu übertragenen Bytes der Langform ausreichen würde?

Anmerkungen zur Kodierung in BER: Längenfeld

Das Längenfeld gibt an, wie viele Datenoktetts folgen. Es können verschiedene Formen unterschieden werden:

- Kurzform: Die Werte liegen zwischen 0 und 127.
- Langform: Das MSB (Most Significant Bit) wird auf 1 gesetzt und die Bits 7 bis 1 kodieren die Anzahl nachfolgender Oktette (127 ist reserviert für eventuelle Erweiterungen).
- Unbestimmte Form: Es werden solange Oktetts übertragen bis der Delimiter (zwei auf Null gesetzte Oktette) verschickt werden. Es ist darauf zu achten, dass diese Struktur nicht in den zuversendenden Daten erscheint (s. Character Stuffing, Kapitel 2).

Frage: Wieviele Byte können mit Hilfe der jeweiligen Längenfelder übertragen werden?

- Mit Hilfe der Kurzform: maximal 127 Byte. Die Langform erlaubt $126 * 8 = 1008$ Bit zur Kodierung der Länge: $2,47 * 10^{303}$ Byte

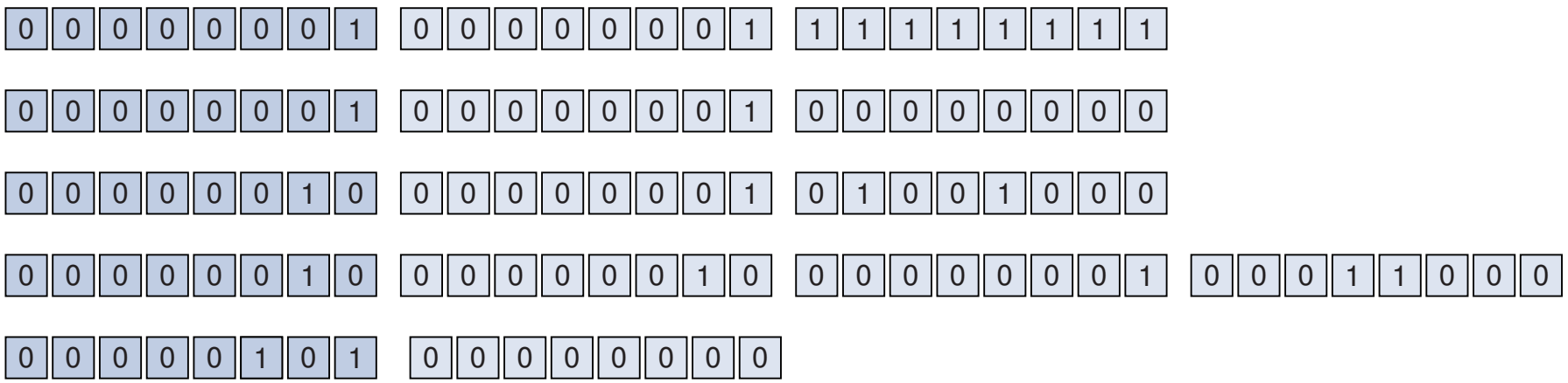
Frage: Welchen Vorteil bringt eine unbestimmte Form, wenn die Anzahl der zu übertragenen Bytes der Langform ausreichen würde?

- Die genaue Länge muss zu Beginn des Sendevorgangs nicht bekannt sein und die Übertragung kann beginnen bevor die genaue Anzahl der Elemente bekannt ist.

BER: Beispiele

Es folgen einige Beispiele umd die Basic Encoding Rules zu verdeutlichen:

- Boolean: TRUE und FALSE
- Integer: 72 und 280
- NULL



- Eine weitere konkrete Transfersyntax stellen die Packed Encoding Rules (PER) dar.
- Sie stellen die Nachfolgeregelung der BER dar und sind wesentlich effizienter vor allem in Bezug auf die Anzahl der übertragenden Bytes.
- Sie werden in den ITU-T X.691 und ISO 8825-2 spezifiziert.
- Praktischen Einsatz finden ASN.1 und die BER im Simple Network Management Protocol (SNMP).
- ASN.1 wird zur Beschreibung der SNMP-Pakete verwendet. Die Kodierung für den Transport wird mit Hilfe der BER erstellt.

Übersicht

Anwendungsschicht

Die Anwendungsschicht (Application Layer) stellt Anwendungen spezifische Dienste sowie Zugang zu Diensten darunter liegender Schichten zur Verfügung. Protokolle der Anwendungsschicht können Teil einer Anwendung sein (z.B. bei Webserver oder Webbrowser).

Beispiele für Dienste der Anwendungsschicht:

- Namensauflösung (DNS)
- Dateitransfer (HTTP, FTP)
- Nachrichtentransfer (SMTP, POP, IMAP, NNTP)
- Entfernte Anmeldung (Telnet, SSH)

Im Folgenden behandeln wir DNS als ein wichtiges Protokoll der Anwendungsschicht etwas genauer.

Domain Name System (DNS)

Motivation:

- Möchte ein Nutzer (Mensch) einen Computer adressieren, will er sich gewöhnlich nicht dessen IP-Adresse merken müssen.
- Stattdessen kann man zur Identifikation eines Zielrechners einen hierarchisch aufgebauten Namen verwenden, z. B. www.google.com.

Domain Name System (DNS)

Motivation:

- Möchte ein Nutzer (Mensch) einen Computer adressieren, will er sich gewöhnlich nicht dessen IP-Adresse merken müssen.
- Stattdessen kann man zur Identifikation eines Zielrechners einen hierarchisch aufgebauten Namen verwenden, z. B. www.google.com.

Domain Name System (DNS):

- Ein DNS-Name (Fully Qualified Domain Name, FQDN) besteht aus einem Hostnamen und einem Suffix:

Domain Name System (DNS)

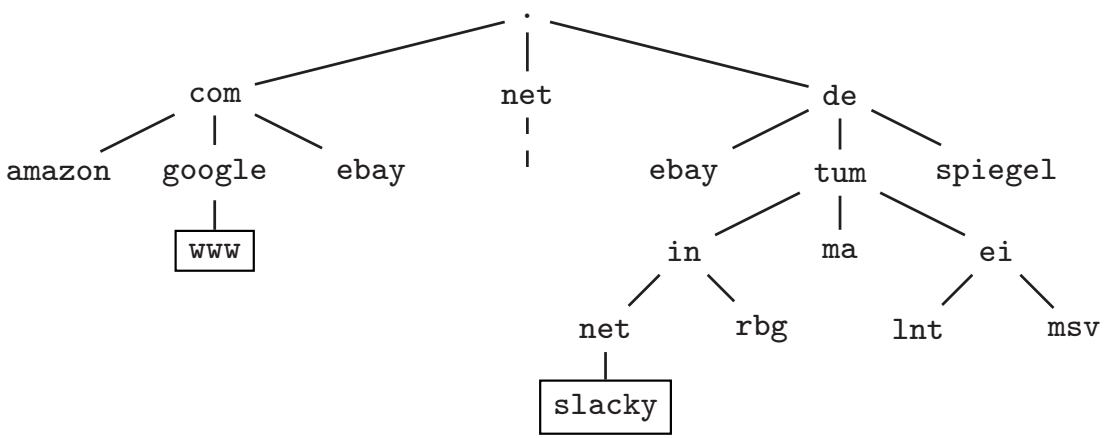
Motivation:

- Möchte ein Nutzer (Mensch) einen Computer adressieren, will er sich gewöhnlich nicht dessen IP-Adresse merken müssen.
- Stattdessen kann man zur Identifikation eines Zielrechners einen hierarchisch aufgebauten Namen verwenden, z. B. www.google.com.

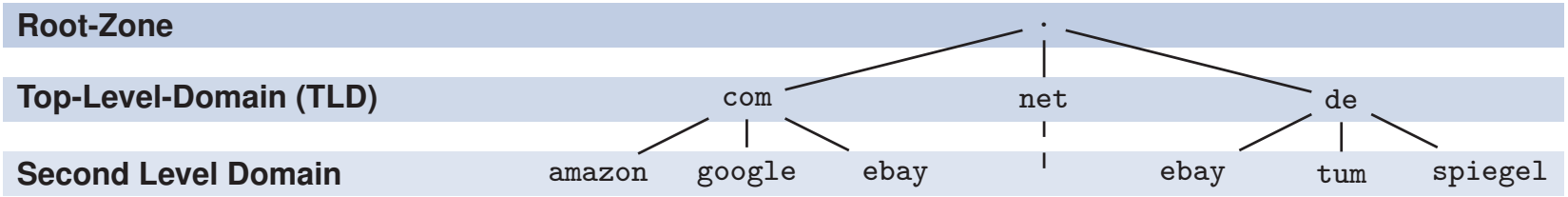
Domain Name System (DNS):

- Ein DNS-Name (Fully Qualified Domain Name, FQDN) besteht aus einem Hostnamen und einem Suffix:

- Das Suffix ist hierarchisch von rechts nach links beginnend bei Root (.) aufgebaut:



Wichtige Begriffe



- Der Namensraum ist in Zonen unterteilt, die unabhängig von einander administriert werden.
- In jeder Zone gibt es einen primären DNS-Server, welcher
  - eine Liste aller Hosts in dieser Zone und
  - eine Liste mit autoritativen DNS-Servern untergordneter Domänen

  verwaltet. Änderungen an den Einträgen geschehen nur über den autoritativen DNS-Server.
- Zusätzlich kann es eine Reihe sekundärer Server geben, welche eine Kopie der Zonendatei besitzen (SZone Transfer", für Lastverteilung und Ausfallsicherheit).
- Die Antworten dieser Server werden als authoritativ bezeichnet und von anderen DNS-Servern gecached.
- Die Cachedauer hängt vom TTL-Wert ab, welcher in jeder Antwort eines Servers enthalten ist.
- Gibt ein DNS-Server einen derartigen gecachten Eintrag weiter, so spricht man von einer nicht-autoritativen Antwort.
- Infolge des Cachings kann es mehrere Stunden (u.U. auch Tage) dauern, bis Änderungen am primären DNS-Server einer Zone im Internet vollständig sichtbar werden.
- Weitere Details zu DNS finden sich u.a. in RFC1035.

In der Zonendatei finden sich DNS Resource Records unterschiedlicher Typen, u.a.:

- SOA – Start Of Authority Enthält Angaben zur Verwaltung der Zone, u.a.
  - Zonenklasse (IN für Internet, HS für Hesiod)
  - Seriennummer
  - Refresh-Abstand (in Sekunden) für Anfragen nach Änderungen
  - Retry-Abstand zur Wiederholung nicht beantworteter Anfragen
  - Expire-Zeit nach der eine Zone als deaktiviert betrachtet wird
  - negativ-Caching-TTL, wenn zu einem angefragten Domainnamen kein Eintrag zugeordnet ist
- A – Hosteintrag IPv4
  Enthält das Mapping zwischen einem Hostnamen und einer IPv4-Adresse
- AAAA – Hosteintrag IPv6
  Enthält das Mapping zwischen einem Hostnamen und einer IPv6-Adresse
- NS – Name Server
  Gibt den FQDN eines DNS-Servers an, welcher für eine (untergeordnete) Domäne authoritativ ist
- CNAME – Common Name
  Ermöglicht es, einem bestehenden Hosteintrag einen weiteren Namen zuzuordnen, z. B. könnte für den Host `typo3.net.in.tum.de.` ein CNAME-Eintrag vorhanden sein, so dass dieser auch unter dem Namen `www.net.in.tum.de.` erreichbar wird.
- MX – Mail Exchanger
  Gibt einen Mailserver (SMTP-Server) für eine Domäne an.

Allgemeine Anmerkungen:

- Einer IP-Adresse können im Prinzip beliebig viele Namen zugewiesen werden.
- Umgekehrt können sich hinter einem Namen auch mehrere Adressen verbergen (einfach mal `nslookup google.com` auf der Kommandozeile ausführen). Warum könnte das sinnvoll sein?
- DNS-Server können auch Reverse-Lookups durchführen, sofern sie dazu konfiguriert wurden (Übersetzung einer IP-Adresse in den/die zugehörigen DNS-Name(n)).

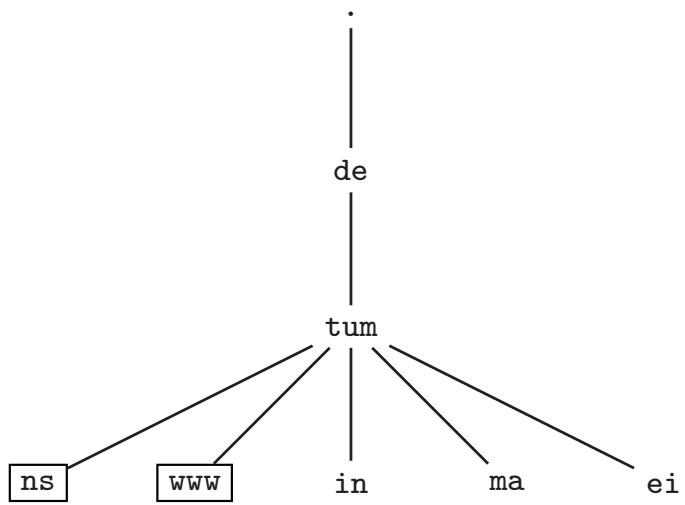
Beispiel: Zonendatei des DNS-Servers `ns.tum.de`

```
$TTL 1W
@  IN  SOA  ns.tum.de. mail.tum.de. (
         2007100801 ; serial
         28800 ; refresh (8 Stunden)
         7200 ; retry (2 Stunden)
         604800 ; expire (1 Woche)
         39600 ; minimum (11 Stunden)
         )

tum.de.         IN  NS  ns.tum.de.
in.tum.de.      IN  NS  ns.in.tum.de.
ma.tum.de.      IN  NS  ns.ma.tum.de.
ei.tum.de.      IN  NS  ns.ei.tum.de.

ns.in.tum.de.   IN  A  <IP von ns.in.tum.de.>
ns.ma.tum.de.   IN  A  <IP von ns.ma.tum.de.>
ns.ei.tum.de.   IN  A  <IP von ns.ei.tum.de.>
ns              IN  A  <IP von ns.tum.de.>
www             IN  A  <IP von www.tum.de.>
```

Ausschnitt aus dem DNS-Namensraum:



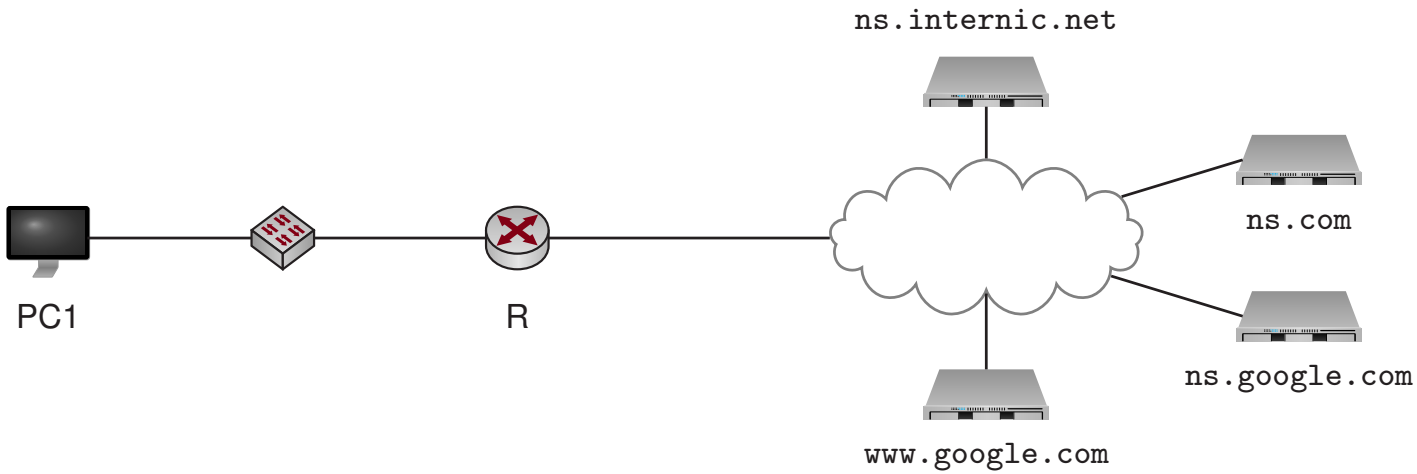
Die Zonendatei von `ns.tum.de.` beinhaltet

- allgemeine Angaben zur Zonendatei (Gültigkeitsdauer, Seriennummer, Refresh-Zeiten usw.),
- eine Liste der authoritativen Namensserver für `tum.de.` selbst sowie alle untergeordneten Domänen, für die `ns.tum.de.` nicht selbst zuständig ist und
- eine Liste mit allen Hostnamen der Domäne(n), für die `ns.tum.de.` selbst zuständig ist.

Wie funktioniert das jetzt aber genau?

Wir nehmen an, PC1 möchte den DNS-Namen `www.google.com` übersetzen. Übliches Setup:

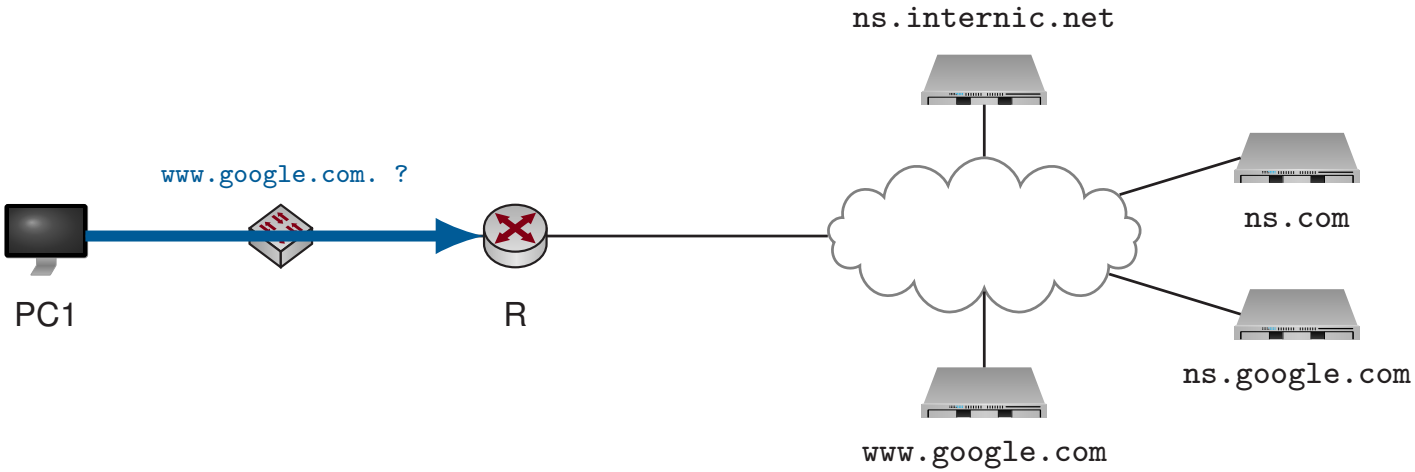
- PC1 ist mit der lokalen IP-Adresse seines Routers R als DNS-Server konfiguriert.
- R selbst ist für keine Zone autoritativ. Seine einzige Aufgabe besteht darin, für PCs im lokalen Netzwerk die Namensauflösung zu übernehmen.



Wie funktioniert das jetzt aber genau?

Wir nehmen an, PC1 möchte den DNS-Namen `www.google.com` übersetzen. Übliches Setup:

- PC1 ist mit der lokalen IP-Adresse seines Routers R als DNS-Server konfiguriert.
- R selbst ist für keine Zone autoritativ. Seine einzige Aufgabe besteht darin, für PCs im lokalen Netzwerk die Namensauflösung zu übernehmen.

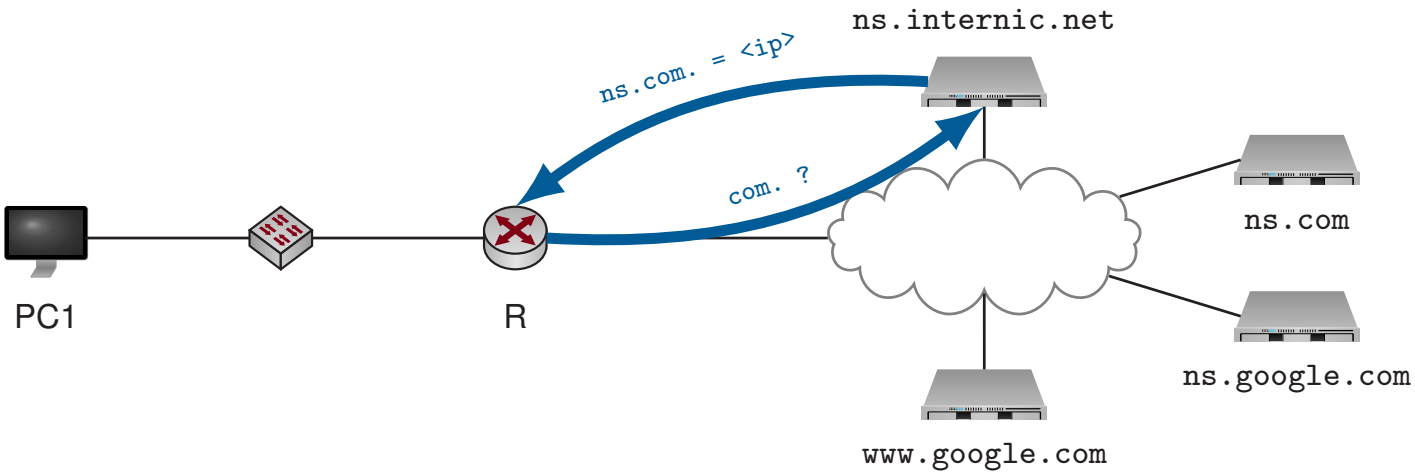


- PC1 schickt an R einen DNS-Request, um `www.google.com` aufzulösen.

Wie funktioniert das jetzt aber genau?

Wir nehmen an, PC1 möchte den DNS-Namen `www.google.com` übersetzen. Übliches Setup:

- PC1 ist mit der lokalen IP-Adresse seines Routers R als DNS-Server konfiguriert.
- R selbst ist für keine Zone autoritativ. Seine einzige Aufgabe besteht darin, für PCs im lokalen Netzwerk die Namensauflösung zu übernehmen.

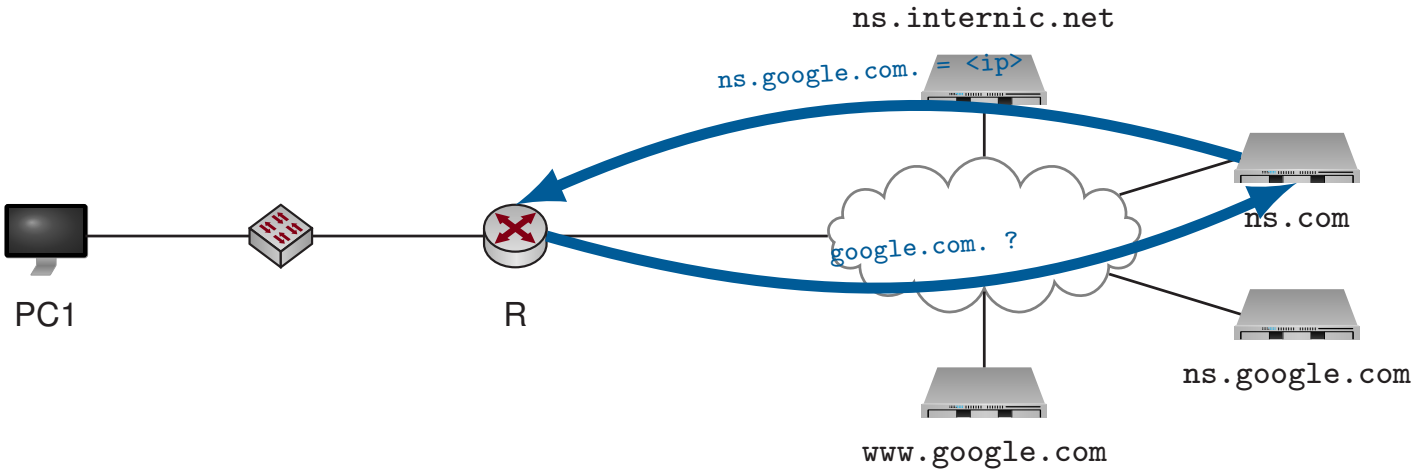


- PC1 schickt an R einen DNS-Request, um `www.google.com` aufzulösen.
- R kennt weder `www.google.com`, noch `google.com` oder `com` selbst und fragt deshalb bei einem der Root-Server an, welche DNS-Server für die Zone `com.` verantwortlich sind.

Wie funktioniert das jetzt aber genau?

Wir nehmen an, PC1 möchte den DNS-Namen `www.google.com` übersetzen. Übliches Setup:

- PC1 ist mit der lokalen IP-Adresse seines Routers R als DNS-Server konfiguriert.
- R selbst ist für keine Zone autoritativ. Seine einzige Aufgabe besteht darin, für PCs im lokalen Netzwerk die Namensauflösung zu übernehmen.

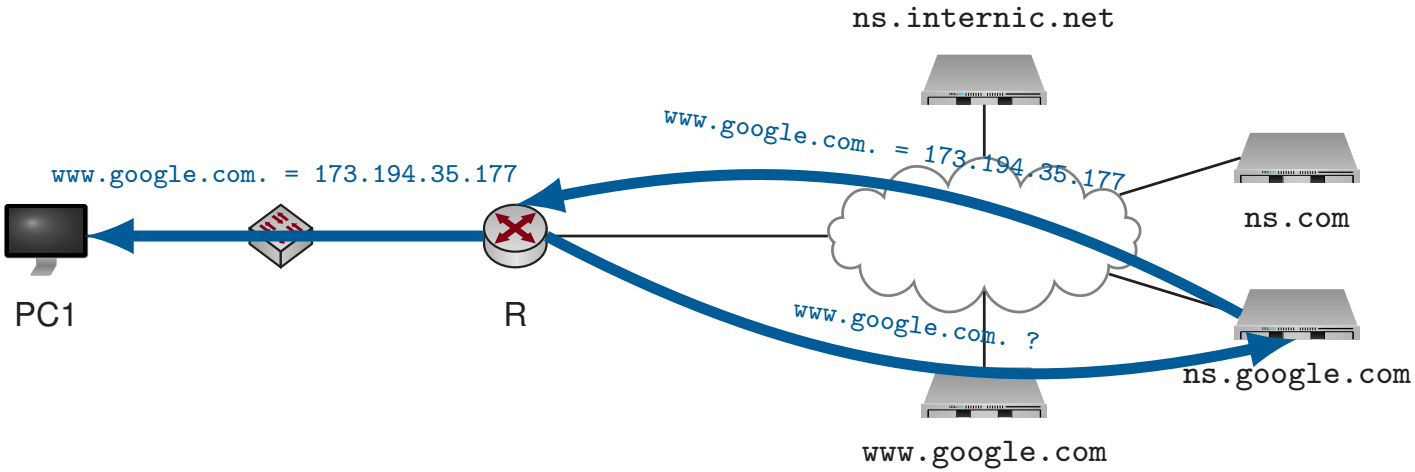


- PC1 schickt an R einen DNS-Request, um `www.google.com` aufzulösen.
- R kennt weder `www.google.com`, noch `google.com` oder `com` selbst und fragt deshalb bei einem der Root-Server an, welche DNS-Server für die Zone `com.` verantwortlich sind.
- Anschließend fragt R bei `ns.com`, wer für `google.com` verantwortlich ist.

Wie funktioniert das jetzt aber genau?

Wir nehmen an, PC1 möchte den DNS-Namen `www.google.com` übersetzen. Übliches Setup:

- PC1 ist mit der lokalen IP-Adresse seines Routers R als DNS-Server konfiguriert.
- R selbst ist für keine Zone autoritativ. Seine einzige Aufgabe besteht darin, für PCs im lokalen Netzwerk die Namensauflösung zu übernehmen.



- PC1 schickt an R einen DNS-Request, um `www.google.com` aufzulösen.
- R kennt weder `www.google.com`, noch `google.com` oder `com` selbst und fragt deshalb bei einem der Root-Server an, welche DNS-Server für die Zone `com.` verantwortlich sind.
- Anschließend fragt R bei `ns.com`, wer für `google.com` verantwortlich ist.
- Schließlich wendet sich R an `ns.google.com` um die IP-Adresse von `www.google.com` aufzulösen. Das Ergebnis wird an PC1 weitergeleitet und im Cache von R gespeichert (TTL ist in der Antwort von `ns.google.com` enthalten).

Rekursive Namensauflösung

- PC1 sendet einmal einen Request an seinen DNS-Server und erwartet als Response das Ergebnis der Namensauflösung.
- Welche Schritte R unternehmen muss (oder unternimmt), um den Request zu beantworten, weiß PC1 nicht.
- Zwischen PC1 und R werden also nur zwei Pakete ausgetauscht.

Iterative Namensauflösung

- Sofern R nicht bereits Teile des angefragten Namens in seinem Cache hat (z. B. den ganzen FQDN oder einen authoritativen DNS-Server für `google.com`), wird der FQDN Schritt für Schritt (iterativ) beginnend bei der Wurzel aufgelöst.
- Die Adressen der DNS-Rootserver hat jeder DNS-Server (also auch der minimalistische Server auf R) gespeichert. Diese bezeichnet man als Root Hints.
- Prinzipiell wäre es natürlich auch möglich (und ist eher die Regel als die Ausnahme), dass R selbst zur rekursiven Namensauflösung konfiguriert ist. In diesem Fall würde R den Request von PC1 einfach an seinen DNS-Server weiterleiten, z. B. den DNS-Server eines Service Providers wie der Deutschen Telekom.